Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Christabella Irwanto

Data-efficient meta-learning with Bayesian deep learning

Master's Thesis
Espoo, October 2, 2020

Supervisor: Assistant Professor Arno Solin

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Christabella Irwanto

Title:
Data-efficient meta-learning with Bayesian deep learning

Date: October 2, 2020 **Pages:** 62

Major: Computer Science **Code:** SCI3042

Supervisor: Assistant Professor Arno Solin

As machine learning is increasingly used in real-world systems, two key methods for function approximation have gained traction: deep learning with neural networks, and Bayesian inference with Gaussian processes (GPs). The two methods have distinct but complementary properties, and Neural Processes (NPs) have emerged as a desirable approach that combines the merits of both.

NPs naturally fall within the meta-learning regime, which is loosely inspired by how humans can learn new tasks from just a few examples, leveraging partially related tasks. As such, we compare NPs to a simple extension of GPs to the meta-learning setting ("Meta-GP"), augmenting the GPs with neural networks via an existing framework known as "deep kernel learning" or "deep mean learning".

Using a few regression benchmarks from both synthetic and real-world one-dimensional data, the Meta-GP outperforms NP in both accuracy and uncertainty calibration. Whether deep kernel or mean learning for the GP is more appropriate, is shown to depend on whether the alternative handcrafted kernel or mean is unsuitable for the data, as expected.

Future work may lie in more rigorous dataset benchmarks, as well as the inclusion of more baseline methods in time series forecasting. It also may be worth considering when it is desirable to forego a valid stochastic process in place of an approximate Neural Process, such as when there is more data, and why a valid stochastic process is desirable for applications.

Keywords: few-shot learning, meta-learning, data-efficient, probabilistic, conditional, neural process

Language: English

Abbreviations and Acronyms

ANP Attentive Neural Process [Kim et al., 2019] 32, 36, 46, 47
AR autoregressive 18, 20, 21, 24
CE calibration error 40, 44, 51
CNN convolutional neural network 9, 10
CNP Conditional Neural Process [Garnelo et al., 2018a] 31, 32, 39, 47
DA domain adaptation 26
DGM deep generative model 5, 6, 17, 18
DKL deep kernel learning [Wilson et al., 2016] 33, 45, 48
ELBO evidence lower bound 18, 21, 39
FSL few-shot learning 5, 28
GAN generative adversarial network 15, 17, 18, 20, 21
GP Gaussian process 5, 6, 8, 10–15, 17, 26, 29, 30, 33–37, 39, 41–45, 47, 48, 50, 51
GPR Gaussian process regression 10, 11, 25, 33, 34, 51
HBM hierarchical Bayesian model 28
LML log marginal likelihood 13, 25, 48
MADE Masked Autoencoder Distribution Estimator [Germain et al., 2015] 24
MAF Masked Autoregressive Flow [Papamakarios et al., 2017] 24
MAML Model-Agnostic Meta-Learning [Finn et al., 2017] 26, 28
ML machine learning 5, 7, 26, 28, 34
MLE maximum likelihood estimation 13
MLP multilayer perceptron 8
MOGP multioutput Gaussian process 10
MTL multi-task learning 26, 27
NAS neural architecture search 26, 28
NF normalizing flow 18, 20–24
NLL negative log-likelihood 40
NN neural network 5, 6, 8–10, 13–15, 17, 20, 25–27, 29, 33–36, 44, 45, 47, 48, 51
NP Neural Process [Garnelo et al., 2018b] 14, 26, 28–30, 32, 34–36, 39, 41–44, 46–48, 51
RBF radial basis function 11, 12
RL reinforcement learning 6, 27, 28
RMSE root-mean-square error 40, 44, 47
RNN recurrent neural network 9
SE squared exponential 33, 35–37, 45, 47, 48, 51
SGD stochastic gradient descent 10, 13, 17, 18, 39
TL transfer learning 26, 27
VAE variational autoencoder 7, 15–18, 20, 21

Contents

Chapter 1

Introduction

Machine learning (ML)—computer algorithms that improve automatically through experience—underpins a vast range of products and services used by millions of people, in areas spanning recommendation engines, traffic prediction, personalized search, fraud detection, and voice assistants. Many of these are driven by *deep learning*, or the use of deep and increasingly complex neural networks (NNs), which have achieved the state of the art in many applications (see Section 2.2). However, deep learning's success typically hinges on the availability of large-scale labeled data and high computational power (see Section 2.2).

As such, the past few years has seen a surge of interest in *meta-learning* or *learning-to-learn*, which holds the promise to alleviate many of the main criticisms of contemporary deep learning such as the aforementioned data and computational efficiency.

Meta-learning has had a long history in cognitive science. Humans can learn from a rich ensemble of partially related tasks, extracting shared information from them and applying that on new tasks [Lake et al., 2016]. If the new tasks have only a few samples or "*shots*", this is called the *few-shot learning (FSL)* problem. For example, children can learn how to use a new word from just a one or a few samples, by leveraging prior knowledge from previously learnt words [Xu and Tenenbaum, 2007]. Another example is how humans can quickly learn the concept of a "Segway" from just one picture, using prior knowledge of related concepts such as motorcycles and scooters Lake et al. [2015].

As such, in ML, meta-learning is often about *learning across many related tasks in order to generalize to new tasks* with potentially limited data. This is useful in many real-world scenarios (see Table 1.1) such as recommendation systems, where each user is interpreted as a "task" and we would like to "learn how to learn" user preferences from many user profiles, in order to generalize to new users with potentially very few ratings or likes.

This thesis focuses on a particular Bayesian flavour of meta-learning, by combining the expressive power of NNs with the data-efficiency of Gaussian processes (GPs), two key ML approaches. Chapter 2 (*"Background"*) surveys and dives into the workings of NNs, GPs, deep generative models (DGMs), and

Scenario	**Task**	**Samples**
Recommendation systems	User profile	User interactions (e.g. likes or ratings) of each user
3D object reconstruction	3D object	Views of each object as seen from different angles
Image completion/inpainting	Image	Seen pixels from each image
Maze navigation with a reinforcement learning (RL) agent	Maze	Episodes of interaction with each maze [Mishra et al., 2017]

Table 1.1: Real-world multi-task meta-learning **scenarios** that illustrate how learning across many related **tasks** helps in generalizing to new tasks with potentially very few **samples**.

meta-learning. Chapter 3 (*"Methods"*) then describes several ways to blend NNs and GPs for meta-learning, borrowing techniques from DGMs. Chapter 4 (*"Experiments"*) uses these methods to conduct experiments, and Chapter 5 (*"Discussion"*) evaluates the experiments' outcomes. Chapter 6 (*"Conclusion"*) rounds off with a summary and directions for future work.

Chapter 2

Background

Function approximation is a core problem in ML. This chapter starts by discussing and comparing two leading approaches to function approximation in the context of supervised learning (Section 2.1), deep neural networks (Section 2.2) and Gaussian processes (Section 2.3), to shed light on how we might combine desirable properties of both (Section 2.4).

Next, Section 2.5 to 2.8 introduces generative modeling, motivates their usefulness in many applications (Section 2.5), and provides a review of deep generative models (Section 2.7) including variational autoencoders (VAEs) and normalizing flows (Section 2.8).

Finally, Section 2.9 introduces meta-learning as a way to improve the data-efficiency of learning algorithms, and shows how stochastic processes can be interpreted in the meta-learning framework for few-shot function estimation.

2.1 Conventional supervised learning

Consider a supervised ML problem (Figure 2.1) with a training dataset $\mathcal{D} = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^n = \{\boldsymbol{X}, \boldsymbol{Y}\}$ of n inputs $\boldsymbol{x}_i \in \mathcal{X}$ and outputs $\boldsymbol{y}_i \in \mathcal{Y}$. We wish to learn some good predictive approximating function(s) $f_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by θ, which yield(s) predictions $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = f_\theta(\boldsymbol{x}_i)$ that are close to the ground-truth outputs $\boldsymbol{y}_i$. The optimal model parameters $\hat{\theta}$ are found by minimizing a loss function $\mathcal{L}_\theta(\mathcal{D}; \theta)$ on the training data $\mathcal{D}$. Since we wish to ultimately make predictions for unseen test inputs, we measure the generalization of the trained predictive model $f_{\hat{\theta}}$ by some evaluation metric(s) on a held-out test dataset $\mathcal{D}_* = \{\boldsymbol{X}_*, \boldsymbol{Y}_*\}$.

In essence, supervised learning is the problem of automatic predictive function approximation from empirical data, since the true underlying function is unknown. The task is called *classification* when the outputs $\boldsymbol{Y}$ are discrete, and *regression* when $\boldsymbol{Y}$ are continuous (for some examples, see Rasmussen and Williams [2005, Chapter 1]). We later contrast this conventional supervised learning setting to meta-learning (Section 2.9).

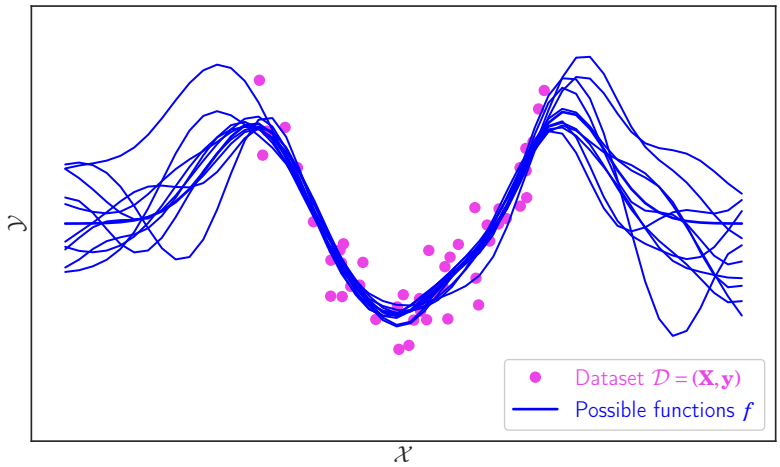


Figure 2.1: Function approximation for supervised learning on data with 1-dimensional inputs and outputs. For illustration, we fit the data using a zero-mean GP with a squared-exponential kernel, and plot the posterior mean as well as 10 samples from the GP posterior.

Two popular approaches to function approximation in machine learning have been i) deep NNs, and ii) Bayesian inference on stochastic processes, most commonly a GP. With this supervised learning regime in mind, we will compare using a NN versus using a GP.

2.2 Neural networks

Deep learning refers to the composition of multiple non-linear processing layers to automatically learn increasingly higher-level representations of data [LeCun et al., 2015]. This composition of layers is central to the success of NN as it provides a particular type of relational inductive bias—that of *hierarchical processing* [Battaglia et al., 2018]. With enough such "building blocks", very complex functions $f_\theta : \mathcal{X} \to \mathcal{Y}$ can be learned.

Perhaps the most ubiquitous deep learning architecture is the **feedforward neural network** or multilayer perceptron (MLP), where the "building blocks" are fully connected layers [Rosenblatt, 1958]. Figure 2.2 illustrates a simple 3-layer feedforward NN. Neurons in adjacent layers are connected by weights (represented by blue arrows), which are the network parameters θ. Each neuron contains a nonlinearity, which enables functions of increasing complexity. The weights are learned automatically from data by optimizing some loss function between the network's prediction $\hat{\boldsymbol{y}}$ and the true output $\boldsymbol{y}$. This is commonly

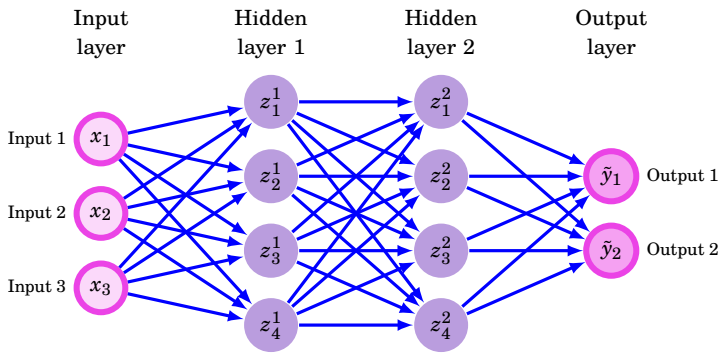


Figure 2.2: A 3-layer feedforward neural network mapping input $\boldsymbol{x} \in \mathbb{R}^3$ to predicted output $\hat{\boldsymbol{y}} \in \mathbb{R}^2$, with two hidden layers of four neurons each (in purple). Nodes i and j in adjacent layers are connected by tunable weights w_{ij} (blue arrows). Each hidden node j (in purple) computes a weighted sum over its inputs from the previous layer, u_i for $i \in \text{in}(j)$, and adds a bias b_j to produce $\sum_{i \in \text{in}(j)} w_{ij} u_i + b_j$. This is then passed through a nonlinear activation function σ to produce an output $\sigma(\sum_{i \in \text{in}(j)} w_{ij} u_i)$ for the next layer. The output nodes act similarly, but can use identity for the activation function.

achieved with gradient descent via backpropagation [Goodfellow et al., 2016, Chapter 6.5] of loss gradients through the network.

Advantages

NNs are powerful universal approximators, capable of approximating any measurable function [Hornik et al., 1989]. They excel at recognizing patterns in the data, automatically performing feature engineering and learning multiple levels of abstraction from raw data, thanks to the relational inductive bias of hierarchical processing.

In fact, the various types of building blocks used in NNs also carry various relational inductive biases. However, the implicit relational inductive bias in a fully connected layer is very weak, since all input units can interact to determine any output unit's value [Battaglia et al., 2018]. If we instead compose convolutional layers to form convolutional neural networks (CNNs), we get a bias for locality and translation invariance; recurrent layers in a recurrent neural networks (RNNs) impose temporal invariance and locality. NNs such as CNNs and RNNs have proven to be effective in achieving the state of the art in many contexts, including visual object recognition [Krizhevsky et al., 2012; He et al., 2015], natural language understanding [Devlin et al., 2019], speech recognition [Hinton et al., 2012], machine translation[Cho et al., 2014; Sutskever et al., 2014], and playing complex games such as Go [Silver et al., 2016, 2018].

Furthermore, deep neural networks are often learned using various descen-

dants of stochastic gradient descent (SGD) [Bottou et al., 2018], which when combined with recent advancements in hardware computation, have enabled efficient learning on large-scale datasets.

Disadvantages

Nevertheless, these well-known successes of deep learning have been achieved within the limited context of supervised learning, where there is **plentiful labeled data** and accordingly **powerful computing resources**.

NNs tend to require a large amount of training data; it has been shown that the volume of training data has a clear logarithmic relationship with the performance of large CNNs on supervised computer vision tasks [Sun et al., 2017], and training deep models with small datasets from scratch or even fine-tuning is likely to result in overfitting or even non-convergence. One reason is that prior knowledge of the underlying function can only be specified in rather limited ways, such as through the NN architecture design. In Section 2.9 and further, we will explore how meta-learning provides an alternative where prior knowledge of the underlying function can be learnt from other functions sampled from a shared process which underlies all functions, and more flexibly encoded in a shared model. Meta-learning is required because a straightforward retraining of a NN on a new task often results in the loss of knowledge of the previously learnt task – a phenomenon known as *catastrophic forgetting*.

As a result of large datasets, training neural networks via gradient descent is also often computationally demanding, thus excluding applications where compute resources are unavailable, such as in edge computing on mobile devices and home appliances et cetera. since it requires the optimization of highly non-convex objective functions

The learnt model parameters are also **difficult to interpret**, especially in large networks, which can lead to models with 'Clever Hans'-type task-solving strategies that are based on spurious correlations in the training data, and are not valid or desirable from a human point of view [Lapuschkin et al., 2019; Zech et al., 2018]

2.3 Gaussian processes

Another solution to the general problem of supervised function approximation as defined in Section 2.1 is GP regression or classification. In this section, we focus on Gaussian process regression (GPR) [Rasmussen and Williams, 2005, Chapter 2]; for the more demanding problem of GP classification, refer to Rasmussen and Williams [2005, Chapter 3].

A GP is a type of stochastic process, that is, a collection of random variables continuously indexed e.g. by time or space, $\{f(\boldsymbol{x}_i)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$. GPs are usually formulated as having a scalar (one-dimensional) output y, although multiple interacting outputs can be considered as well with multioutput Gaussian processes

(MOGPs) [van der Wilk et al., 2020b]. As such, according to the *function space view* [Rasmussen and Williams, 2005], a (scalar) GP defines a distribution over functions $p(f)$ where $f : \mathbb{R}^{d_x} \to \mathbb{R}$, such that for any finite subset $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, the marginal joint distribution over their function evaluations $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))$ has a multivariate Gaussian distribution,

$$p(\boldsymbol{f} \mid \boldsymbol{X}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{\mu}, \boldsymbol{K}), \tag{2.1}$$

where $\boldsymbol{\mu} = (m(\boldsymbol{x}_1), ..., m(\boldsymbol{x}_N))$ and $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The mean function m is the average over all function values at $\boldsymbol{x}$,

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})], \tag{2.2}$$

and is commonly left as $m(\boldsymbol{x}) = 0$ since GPs are often flexible enough to model the mean arbitrarily well [Murphy, 2013]; this also helps to simplify posterior computations and only do inference via the covariance function κ [Schulz et al., 2018]. However, a non-zero mean GP is important in some applications [Dunlop et al., 2018].

The covariance function κ is a positive definite kernel function that models the dependence between function values at different inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \operatorname{cov}\left(f(\boldsymbol{x}), f(\boldsymbol{x}')\right) = \mathbb{E}\left[\left(f(\boldsymbol{x}) - m(\boldsymbol{x})\right)\left(f(\boldsymbol{x}') - m(\boldsymbol{x}')\right)\right], \tag{2.3}$$

and consequently determines the shape of functions from the GP. A popular GP kernel is the radial basis function (RBF) kernel, for modeling smooth and stationary functions. In short, we assume our latent function of interest $f(\boldsymbol{x})$ is distributed as a GP parameterized by its mean and kernel functions, m and κ:

$$f \sim \mathcal{GP}(m, \kappa). \tag{2.4}$$

In GPR, we also assume that observations $\boldsymbol{Y}$ have a **factorized Gaussian likelihood**, since we assume an underlying model $\boldsymbol{Y} = \boldsymbol{f} + \boldsymbol{\epsilon}$ with identically distributed Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$. Note that the likelihood's mean itself, $\boldsymbol{f} = f(\boldsymbol{X})$, is also a random variable since f is drawn from a GP (Equation 2.4):

$$\boldsymbol{Y} \mid \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma_y^2 \boldsymbol{I}). \tag{2.5}$$

The joint distribution of observed training outputs $\boldsymbol{Y}$ and test predictions $\boldsymbol{f}_*$ under the prior is [Rasmussen and Williams, 2005, Equation 2.21]

$$\begin{pmatrix} \boldsymbol{Y} \\ \boldsymbol{f}_* \end{pmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{pmatrix} \boldsymbol{K}_y & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{pmatrix}\right), \tag{2.6}$$

where $\boldsymbol{K}_y = \kappa(\boldsymbol{X}, \boldsymbol{X}) + \sigma_y^2 \boldsymbol{I}$, $\boldsymbol{K}_* = \kappa(\boldsymbol{X}, \boldsymbol{X}_*)$, and $\boldsymbol{K}_{**} = \kappa(\boldsymbol{X}_*, \boldsymbol{X}_*)$.

To get the posterior distribution over functions we need to restrict this joint prior distribution to contain only those functions which agree with the observed data points, that is, condition the joint prior on the observations $\boldsymbol{Y}$ to give a

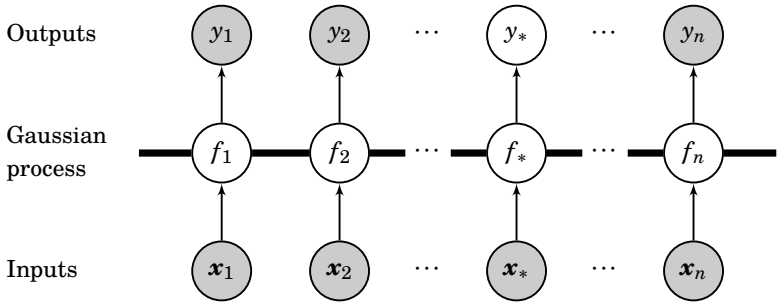


Figure 2.3: Graphical model of a GP for regression, with n training input-output pairs $(\boldsymbol{x}_i, y_i)_{i=1}^n$, a test-time input and output, $\boldsymbol{x}_*$ and y_*, and latent variables f. Shaded variables are observed while unshaded variables are unobserved. The thick horizontal bar represents a set of fully connected nodes. Diagram modified from Rasmussen and Williams [2005, Figure 2.3].

Gaussian **posterior predictive distribution** for noise-free test outputs $\boldsymbol{f}_*$ given new inputs $\boldsymbol{X}_*$ [Rasmussen and Williams, 2005, Equations 2.22–2.26],

$$p(\boldsymbol{f}_* \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{Y}) = \mathcal{N}\left(\boldsymbol{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*\right), \tag{2.7}$$

where the parameters of the posterior predictive are

$$\begin{aligned}\boldsymbol{\mu}_* &= \boldsymbol{K}_*^\top \boldsymbol{K}_y^{-1} \boldsymbol{Y}, \\ \boldsymbol{\Sigma}_* &= \boldsymbol{K}_{**} - \boldsymbol{K}_*^\top \boldsymbol{K}_y^{-1} \boldsymbol{K}_*.\end{aligned}$$

To use a deterministic non-zero mean function $m(\boldsymbol{x})$, apply the zero mean GP to the difference between the observations and $m(\boldsymbol{x})$ such that the posterior predictive mean becomes [Rasmussen and Williams, 2005, Equation 2.37]

$$\boldsymbol{\mu}_* = m(\boldsymbol{X}_*) + \boldsymbol{K}_*^\top \boldsymbol{K}_y^{-1} (\boldsymbol{Y} - m(\boldsymbol{X})), \tag{2.8}$$

while the variance remains unchanged.

We can simply compute the predictive distribution of noisy test targets $\boldsymbol{Y}_*$ by adding noise variance $\sigma_y^2 I$ to the predictive covariance of $\boldsymbol{f}_*$:

$$p(\boldsymbol{Y}_* \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{Y}) = \mathcal{N}\left(\boldsymbol{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_* + \sigma_y^2 I\right). \tag{2.9}$$

Marginal likelihood

The trainable hyperparameters θ of the GP (e.g. length scales of the RBF kernel, or parameters of the mean and kernel functions in general) are most commonly optimized with a general-purpose gradient-based method, such as L-BFGS-B

or SGD, to maximise the *log marginal likelihood (LML)*, which is given by the integral of the likelihood (Equation 2.5) times the prior (Equation 2.1),

$$\mathcal{L}_\theta(\mathcal{D};\theta) = \log p(\boldsymbol{Y} \mid \boldsymbol{X}) \tag{2.10}$$

$$= \log \int p(\boldsymbol{Y} \mid \boldsymbol{f}, X) p(\boldsymbol{f} \mid X)\, \mathrm{d}\boldsymbol{f} \tag{2.11}$$

$$= \log \mathcal{N}\left(\boldsymbol{Y} \mid \boldsymbol{0}, \boldsymbol{K}_y\right) \tag{2.12}$$

$$= -\frac{1}{2}(\boldsymbol{Y} - m(\boldsymbol{X}))^T \boldsymbol{K}_y^{-1}(\boldsymbol{Y} - m(\boldsymbol{X})) - \frac{1}{2}\log\left|\boldsymbol{K}_y\right| - \frac{N}{2}\log(2\pi). \tag{2.13}$$

The term marginal likelihood refers to marginalization over the function values $\boldsymbol{f}$. Using a maximum likelihood estimation (MLE) method in this way, to optimize model hyperparameters instead of parameters, is also called type II maximum likelihood (ML-II) approximation [Rasmussen and Williams, 2005].

Advantages

Inference with GPs for regression have closed-form solutions; the marginal likelihood for learning optimal hyperparameters is given by Equation 2.13), and the posterior predictive for making predictions at new points is given by 2.9.

Furthermore, one can design the GP's covariance kernel to control the inductive biases of the model, such as whether we expect the functions to be smooth, periodic, or have conditional independence properties. Different kernels can be combined to create a rich set of interpretable and reusable building blocks [Duvenaud et al., 2013].

Finally, as a non-parametric method, the information capacity of a GP grows with the amount of available data, but its complexity is automatically calibrated by maximizing its marginal likelihood, without the need for regularization or cross-validation like NNs [Rasmussen and Williams, 2005].

Disadvantages

Although exact inference with GPs is an advantage, in practice, GPs are computationally infeasible for large dataset sizes. The closed-form equations (2.13 and 2.9) require the inversion and determinant of kernel matrices with sizes corresponding to the dataset size n, which is naively $\mathcal{O}(n^3)$.

GPs also require a Gaussian likelihood (shown in Equation 2.5) that is conjugate to the GP prior for closed-form inference, which is often unrealistic.

Approximate inference methods exist that address those two problems, with approximations to the posterior [van der Wilk et al., 2020a] or approximations to the model [Quiñonero-Candela and Rasmussen, 2005], but the dominant methods at best scale quadratically (e.g. Snelson and Ghahramani [2005]).

Lastly, the identification of the kernel and its hyperparameters heavily influences the resulting family of functions and predictive distrtibutions of GPs. This

	Neural networks	**Gaussian processes**
1	✓ (Finite) **adaptive** basis functions	✗ (Infinite) **fixed** basis functions
2	✓ Fast, scalable inference	✗ Slow inference of $\mathcal{O}(N^3)$
3	✗ Needs a lot of training data	✓ Data-efficient
4	✗ Learns a single function with no uncertainty quantification	✓ Learns a distribution over functions, quantifies uncertainty

Table 2.1: High-level comparison of NNs and GPs for function approximation with supervised learning.

is usually an additional step that requires manual intervention, potentially requiring model selection to compare a range of kernel forms, and hyperparameter search.

2.4 Neural networks versus Gaussian processes

To summarize, the complementary properties of neural networks and Gaussian processes (see Table 2.1) have motivated considerable work in combining the two, of which some approaches will be later covered in Chapter 3.

Property 1 of Figure 2.1 is that NNs are powerful universal approximators that can flexibly learn basis functions from the data directly [MacKay, 1998]. On the other hand, GPs traditionally use predefined kernels in restricted functional forms, which requires human intervention in the choice and tuning of the kernel. This has motivated methods for creating expressive GP kernels which can automatically discover rich structure in data with the help of NNs, such as GPs with deep mean and/or kernel functions [Fortuin and Rätsch, 2019; Wilson et al., 2016; Rothfuss et al., 2020], effectively resulting in GPs with infinitely many *adaptive* basis functions. Taking it a step further, methods such as Neural Processes (NPs, Garnelo et al. [2018b]) learn not just an implicit kernel but an implicit stochastic process directly from the data. These methods will be covered in more detail in the following Chapter 3.

Property 2 of Figure 2.1 is another advantage of NNs. At test-time, running forward-passes through NNs is computationally efficient. On the other hand, exact inference in GPs scales cubically for large datasets. Nevertheless, the training of a NN can be expensive and challenging.

A disadvantage of NNs (Property 3 of Figure 2.1) is that the success of NNs (as described in Section 2.2) tends to be within the regime of supervised learning on a *large amount of annotated data* that is time-consuming and expensive to acquire. This also results in a computationally intensive training phase. Whereas GPs are more data-efficient since knowledge of the underlying function is already encoded in the prior stochastic process even before observing any data, albeit in limited ways. NN-based **meta-learning** methods (Section 2.9) provide a data-

driven alternative to hand engineering priors. Generative models (Section 2.5) can also help to alleviate this limitation by enabling **unsupervised learning** on unlabelled data.

Finally, NNs are deterministic models and can only yield point estimates, unlike GPs, which are probabilistic models that can provide uncertainty estimates. An implication of this is that NNs need to be trained from scratch for new functions, whereas GPs allow *reasoning about multiple functions*.

2.5 Generative models

Both NNs and GPs, when used in the manner described in Sections 2.2 and 2.3 for supervised function approximation, are **discriminative models**. Discriminative approaches directly compute input-output mapping $f : \mathcal{X} \to \mathcal{Y}$ for classification and regression, or in the case of probabilistic models, predictive conditional distribution $p(\boldsymbol{Y} \mid \boldsymbol{X})$ [Jebara, 2004].

Concretely put, the NN discriminative model directly learns a network f_θ by optimizing parameters θ on the training dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{Y}\}$, and obtains predictions on the test set of inputs $\boldsymbol{X}_* = (\boldsymbol{x}_*)_{i=1}^n$ through forward-passes $\boldsymbol{y}_* = f_\theta(\boldsymbol{x}_*)$, while the GP discriminative model directly learns a predictive conditional distribution of test outputs $\boldsymbol{Y}_* = (y_*)_{i=1}^n$ given the test inputs $\boldsymbol{X}_*$ as well as the training dataset $\mathcal{D}$, given by the GP posterior predictive distribution $p(\boldsymbol{Y}_* \mid \boldsymbol{X}_*, \mathcal{D})$ (Equation 2.9).

This is a counterpoint to **generative models**, which are traditionally described as learning the joint distribution $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x} \mid \boldsymbol{y})p(\boldsymbol{y})$ [Jebara, 2004]—or in the case of unsupervised learning on unlabelled datasets, $p(\boldsymbol{x})$.

However, we adopt a broader definition of generative models to encompass *any model that learns the distribution of data, and thus are able to* ***generate*** *new data samples*. This is because the traditional definition does not strictly hold for many contemporary approaches, e.g. generative adversarial networks (GANs) and VAEs, which nevertheless are commonly referred to as "generative models" since they are used to generate samples. For example, implicit generative models such as GANs are sampling-based and do not explicitly model a probability density in the first place (as illustrated in Figure 2.5b); GANs entirely avoid the intractable learning of a generative model $p(\boldsymbol{x})$ by instead learning a divergence measure via the discriminator network.

2.6 Why generative models?

Generative models find wide applicability both as an end in themselves, as well as in downstream applications such as semi-supervised learning and meta-learning—the latter of which is the focus of this thesis (as described in Section 2.9).

As an end in themselves, generative models find myriad uses in, for example,

1. Drawing new samples $\boldsymbol{x} \sim p(\boldsymbol{x})$, e.g. for image captioning [Vinyals et al., 2015; Xu et al., 2015], machine translation [Cho et al., 2014; Sutskever et al., 2014], speech synthesis [Oord et al., 2016], and for model checking by sampling.
2. Estimating density $p(\boldsymbol{x})$, e.g. for anomaly detection [Choi et al., 2018], or for count based exploration strategies on challenging reinforcement learning environments [Ostrovski et al., 2017].
3. Representation learning, e.g. for information retrieval, data compression, model interpretability (disentanglement of latent factors). **Representation learning** involves performing unsupervised feature learning on unlabelled data, which is far more plentiful and cheaper to obtain, and applying these "pre-trained" representations to downstream tasks—this is also referred to as "unsupervised pre-training".

Representation learning also enables a popular downstream application of generative models, **semi-supervised learning**. Despite the successes of the supervised deep learning approach detailed in Section 2.2, one large drawback of the supervised approach is the difficulty in obtaining enough labeled data. Semi-supervised learning seeks to improve the data-efficiency of a supervised task by learning from unlabeled examples, which is far more plentiful and cheaper to obtain. This has typically been approached by two complementary methods: **representation learning**—which may use generative models—and label propagation [Hénaff et al., 2019].

Early work in semi-supervised learning was driven by representation learning via generative models; Kingma et al. [2014] apply and extend the VAE to model both labeled and unlabeled data, Odena [2016]; Izmailov et al. [2019] extends GANs by augmenting the discriminator to predict class labels, and Izmailov et al. [2019] uses a normalizing flow trained on unlabeled data directly as a Bayes classifier, extending it with consistency regularization. In natural language understanding, the generative pre-training of a language model has proven highly effective for learning representations that improve many downstream tasks, either with discriminative fine-tuning [Dai and Le, 2015; Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2019] or more recently, even without [e.g. GPT-2; Radford et al., 2019].

However, in computer vision where pixel-level reconstruction is computationally expensive and may not be necessary for representation learning [Hénaff et al., 2019], generative unsupervised pre-training [Donahue and Simonyan, 2019] has been outperformed by discriminative self-supervision which directly formulates tasks on the learned representation, particularly techniques based on contrastive learning in the latent space [Hénaff et al., 2019; Oord et al., 2018].

A complementary approach is **label propagation**, which can be successfully combined with representation learning [Zhai et al., 2019]. A classifier is first trained on a subset of labeled data, then used to label parts of the unlabeled dataset. This label propagation can either be discrete (as in pseudo-labeling) or

continuous (as in entropy minimization). Apart from semi-supervised learning, other downstream applications of generative models include **active learning** and **meta-learning**. *Active learning* [Settles, 2009] is a particular semi-supervised learning problem of selecting informative unlabeled data to query an oracle for its label. Meta-learning is another area where generative models have been compellingly applied to. Particularly, many deep generative latent variable models (optimized with amortized VI) used for meta-learning, such as Neural Processes (Section 3.2), LEO [Rusu et al., 2018], and Versa [Gordon et al., 2019a]), are heavily inspired by the design of the VAE or the generator of the GAN, which we introduce in the following section.

2.7 Deep generative models

DGMs generally refer to probabilistic models with multiple layers of stochastic or deterministic variables [Song and Ou, 2018]. Like NNs and GPs, DGMs are nonlinear function approximators, but DGMs are generative and thus offer additional applications as outlined in Section 2.6.

DGMs, or probabilistic models in general, can be broadly classified into directed and undirected graphical models [Frey and Jojic, 2005; Koller and Friedman, 2009]. Directed models produce normalized probabilities, while undirected models require a normalizing constant. In general, undirected models, also known as "random fields" or "energy-based models", are more challenging to train than directed models because calculating the log-likelihood and its gradient is analytically intractable due to the normalizing constant [Song and Ou, 2018]. For example, deep Boltzmann machines [DBMs, Salakhutdinov, 2015] use both variational and Markov chain approximations for training, evaluation, and sampling, and are impractical for large datasets. As such, in this thesis, we focus on **directed models**, as opposed to undirected models.

Furthermore, as discussed in Section 2.2, successful applications of deep learning tend to involve differentiable models that are trained via gradient descent, which has enabled scaling to high dimensions. Such inference schemes have achieved great progress in generative models of images [Salimans et al., 2016], text [Bowman et al., 2015], and more. Therefore, while there exist deep directed models such as sigmoid belief networks which are trained via Gibbs sampling, or others which rely on other sampling-based inference such as MCMC, or variational inference, in this thesis we focus on approaches that are **end-to-end differentiable** and can be efficiently trained, e.g. with minibatch SGD.

Now that we have narrowed the scope down to **directed**, **differentiable** DGMs, we can further split them (as shown in Figure 2.4) into two categories: **prescribed models** and **implicit models** [Diggle and Gratton, 1984; Mohamed and Lakshminarayanan, 2016].

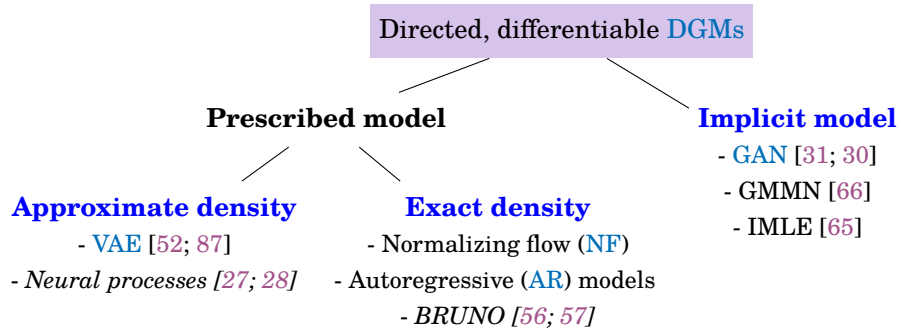


Figure 2.4: A taxonomy of directed, end-to-end differentiable, deep generative models, modified from Goodfellow [2016, Figure 9].

Prescribed models with approximate density

Prescribed models directly model an exact functional form for the data likelihood $p(\boldsymbol{x})$; for example, prescribed latent-variable models such as VAEs (Figure 2.5a) introduces unobserved latent variable(s) $\boldsymbol{z}$ to explain hidden causes generating the data $\boldsymbol{x}$. The task is then to maximize the likelihood over the entire dataset, of which each data point can be expressed as an integral over the latent variables, $p(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}\,|\,\boldsymbol{z})p(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}$. However, this integral is intractable when the conditional distribution $p_\theta(\boldsymbol{x}\,|\,\boldsymbol{z})$ is a neural network, which is why VAEs rely on a **variational approximation** $q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x})$ to the unknown posterior $p(\boldsymbol{z}\,|\,\boldsymbol{x})$, in order to construct a variational lower bound on the marginal likelihood termed as the evidence lower bound (ELBO):

$$p(\boldsymbol{x}) \geq \text{ELBO} = \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - D_{\text{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})). \tag{2.14}$$

In Kingma and Welling [2014], the prior is $p_\theta(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, and neural networks parameterize the conditional distributions $q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}\,|\,\mu_\phi(\boldsymbol{x}), \sigma^2_\phi(\boldsymbol{x})\odot I)$ and $p_\theta(\boldsymbol{x}\,|\,\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}\,|\,\mu_\theta(\boldsymbol{z}), \sigma^2_\theta(\boldsymbol{z})\odot I)$, where $\odot$ signifies the element-wise product. These are the *encoder* (also called the inference network/model, recognition network/model, or the variational distribution) and the *decoder* (or generative model) respectively, as illustrated in Figure 2.5a. The inference network $q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x})$ *amortizes* the cost of variational inference by using the same set of global variational parameters ϕ for all data points. The gradient of the expectation term in 2.14 is computed with the reparameterization trick, while the KL divergence in 2.14 and its gradient can be calculated analytically [Kingma and Welling, 2014]. The ELBO is then maximized using SGD.

Prescribed models with exact density

Another type of prescribed model are those with tractable densities, such as normalizing flows (Figure 2.5c) and autoregressive models (Figure 2.5d). Since

$p(\boldsymbol{x})$ is tractable, such models can be simply trained by maximum likelihood of the training dataset without having to rely on approximations.

Implicit models with exact density

In contrast to prescribed models, **implicit generative models** (also called **likelihood-free** models) directly specify a stochastic procedure with which to generate data (Figure 2.5b). To sample from such a model, we first sample a latent variable $\boldsymbol{z} \in \mathbb{R}^k$ from an analytical prior distribution $p(\boldsymbol{z})$, and apply a deterministic transformation f mapping $\boldsymbol{z}$ to the observed variables $\boldsymbol{x} \in \mathbb{R}^d$, as shown in Figure 2.5b. Implicit models are easy to sample from, but impossible for evaluating likelihood, so it is not feasible for certain applications areas of generative models as outlined in Section 2.5, such as density estimation or representation learning.

Generally, implicit models specify an effective likelihood function $p(\boldsymbol{x})$:

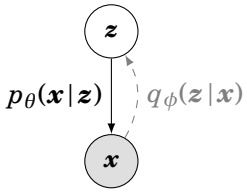
$$p(\boldsymbol{x}) = \frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_d} \int_{\{\boldsymbol{z} \mid f(\boldsymbol{z}) \leq \boldsymbol{x}\}} p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z}, \tag{2.15}$$

since the transformed density $p(\boldsymbol{x})$ is the derivative of the cumulative distribution function of the latent density with respect to $\boldsymbol{x}$. In most popular implicit models such as GANs, the function f is a complex non-linear function with $d > k$, so the integral (2.15) is intractable since it is difficult to for example determine the set $\{\boldsymbol{z} \mid f(\boldsymbol{z}) \leq \boldsymbol{x}\}$ for the integration regions, or to compute the high-dimensional derivative [Mohamed and Lakshminarayanan, 2016].

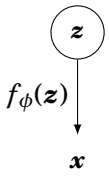
Therefore, implicit models side-step the intractability of the likelihood with likelihood-free methods. GANs do so by optimizing a minimax objective with two neural networks, the discriminator and the generator, competing against each other. However, the lack of an explicit density complicates the quantitative evaluation of implicit models' generalization performance. Typical ad-hoc sample quality metrics such as Inception score for images [Salimans et al., 2016] are prone to failure modes such as memorizing the training data and missing modes [Theis et al., 2016]. Other methods for estimating the likelihoods of GANs, use annealed importance sampling [Wu et al., 2016] and kernel density estimation [Goodfellow et al., 2014] which require expensive Markov chains, and assume a Gaussian observation model which can lead to inaccurate estimates [Grover et al., 2018].

2.8 Normalizing flows

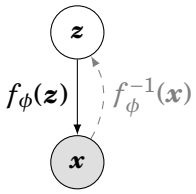
Methods of training implicit models are necessitated by the intractability of the constructed likelihood (Equation 2.15) when the function f is complex. However, if f is well-defined, such as when it is invertible, we recover the familiar rule for transformations of probability distributions.



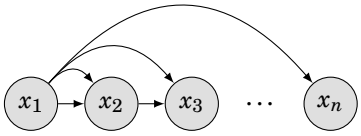
(a) VAE: a **prescribed** latent variable model with **approximate** density. The solid line represents the VAE's decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$, which is the generative model. The dashed line represents the VAE's encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$.



(b) GAN: an **implicit** latent variable model. f_ϕ is a deterministic transformation known as the GAN's generator. The GAN's discriminator is omitted because it is merely an auxiliary object created to approximate an f-divergence.



(c) NF: a **prescribed** latent variable model with **exact** (or **approximate**) density. The transformation f_ϕ is deterministic and invertible. NFs can be thought of as VAEs with Dirac delta distributions on restricted f_ϕ and f_ϕ^{-1}: $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \delta(T(\boldsymbol{z}))$ and $q_\theta(\boldsymbol{z} \mid \boldsymbol{x}) = \delta\left(T^{-1}(\boldsymbol{x})\right)$ [Gritsenko et al., 2019]



(d) AR Bayesian network: a **prescribed** fully-observed model with **exact** density We fix an ordering $x_1, x_2, \ldots, x_n$ and each random variable depends on the preceding random variables. Generally for an AR model, the conditionals $p(x_i \mid \boldsymbol{x}_{<i})$ are specified as parameterized functions such as a NN.

Figure 2.5: Generally, VAEs, GANs, and NFs are mathematically the same object, that is, latent variable models where $\boldsymbol{z}$ is a latent Gaussian random variable pointing to an observed $\boldsymbol{x}$. AR models are fully-observed. The key similarities and differences are outlined in Table 2.2.

NFs, also called invertible density models or flow-based generative models, use the change-of-variables technique to construct flexible probability distributions over continuous random variables. NFs were initially proposed by Rezende and Mohamed [2015] and later thoroughly surveyed by Papamakarios et al. [2019]. Figure 2.6 captures the crux of normalizing flows, which is simply that a real D-dimensional vector $\boldsymbol{z}$ sampled from a simple *base distribution* $p_z(\boldsymbol{z})$ is transformed into a continuous random vector $\boldsymbol{x}$ sampled from a complex distribution of interest, $p_x(\boldsymbol{x})$, via a deterministic invertible and differentiable function T.

The transformation T must be a diffeomorphism, i.e. T is *invertible* and both T and T^{-1} are *differentiable*, so that the change of variables theorem can be used to express the exact likelihood $p_x(\boldsymbol{x})$ in terms of either the Jacobian of T or T^{-1} (and the base distribution):

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(\boldsymbol{z}) \left|\det J_T(\boldsymbol{z})\right|^{-1} = p_{\boldsymbol{z}}(\boldsymbol{z}) \left|\det J_{T^{-1}}(\boldsymbol{x})\right|. \tag{2.16}$$

	VAE	GAN	AR models	NF
Likelihood	Prescribed	Implicit	Prescribed	Prescribed
Likelihood evaluation	Approximate: $\log p(\boldsymbol{x}) \geq$ ELBO	Impossible	Exact: $p(\boldsymbol{x}) = \prod_{i=1}^{d} p(x_i \mid x_{1:i-1})$	Exact[1]: $p(\boldsymbol{x}) = p(\boldsymbol{z})\left\lvert \det J_{T^{-1}}(\boldsymbol{x})\right\rvert$
Training	Approximate inference (variational inference)	Likelihood-free inference (adversarial training)	Maximum likelihood estimation	Maximum likelihood estimation
Latent variables	Yes	Yes	No (fully-observed)	Yes
Posterior inference	Approximate $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$	Impossible (no mechanism)	Impossible (no latent variables)	Exact $\boldsymbol{z} = T^{-1}(\boldsymbol{x})$
Sampling	$\hat{\boldsymbol{z}} \sim p_\theta(\boldsymbol{z})$ $\boldsymbol{x} \sim p(\boldsymbol{x} \mid \boldsymbol{z} = \hat{\boldsymbol{z}})$	$\hat{\boldsymbol{z}} \sim p_\theta(\boldsymbol{z})$ $\boldsymbol{x} = G(\hat{\boldsymbol{z}})$	$x_i \sim p(x_i \mid x_{1:i-1})$ for $i = 1, \ldots, d$	$\hat{\boldsymbol{z}} \sim p_\theta(\boldsymbol{z})$ $\boldsymbol{x} = T(\boldsymbol{z})$

[1] Optimizing a continuous NF on uniformly dequantized discrete data corresponds to maximizing a lower bound on a discrete log-likelihood [Theis et al., 2016].

Table 2.2: High-level comparison of main categories of generative models.

T's invertibility also implies that the dimensionalities of $\boldsymbol{z}$ and $\boldsymbol{x}$ are equal.

Since diffeomorphisms are composable, Figure 2.7 shows how we can build arbitrarily complex transformations T by the **finite composition** of simple discrete transformations $T_K \circ \cdots \circ T_1$, much like neural networks compose building blocks. The term "flow" hence refers to the flow of transformations T that $\boldsymbol{z}$ undergoes, and "normalizing" refers to how the inverse flow T^{-1} is transforming—or in a sense "normalizing"—$\boldsymbol{x}$ into a sample from a prescribed density $\boldsymbol{z}$ [Papamakarios et al., 2019]. Evaluating the likelihood $p_x(\boldsymbol{x})$ then becomes:

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z}) \prod_{k=1}^{K} \left|\det J_{T_k}(\boldsymbol{z}_{k-1})\right|^{-1}, \quad \text{where } \boldsymbol{z} = T^{-1}(\boldsymbol{x}). \tag{2.17}$$

We can implement either T_k or T_k^{-1} as a **neural network** with parameters ϕ, denoted as $f_\phi : \boldsymbol{z} \to \boldsymbol{z}'$, such that it is invertible and has a tractable Jacobian determinant. If f_ϕ does not have an efficient inverse, either density evaluation or sampling will be inefficient depending on whether f_ϕ implements T_k or T_k^{-1}, since the forward transformation T is used when sampling, and inverse transformation T^{-1} is used when evaluating density.

Autoregressive flows are normalizing flows where this network f_ϕ is a strictly monotonic (and therefore invertible) *transformer* τ parameterized by the output of an autoregressive (and not necessarily invertible) *conditioner* c [Papamakarios et al., 2019]:

$$z_i' = \tau(z_i; c_i(\boldsymbol{z}_{<i})) \tag{2.18}$$

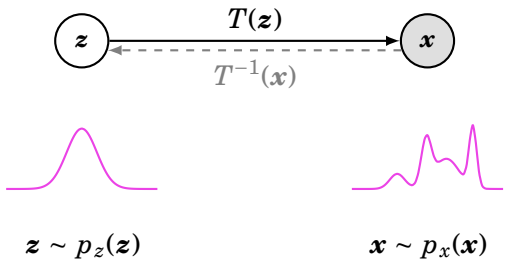


Figure 2.6: NFs allow us to transform a simple distribution $p_u(\boldsymbol{u})$ to a complex distribution $p_x(\boldsymbol{x})$ via a diffeomorphism (invertible and differentiable function) T.

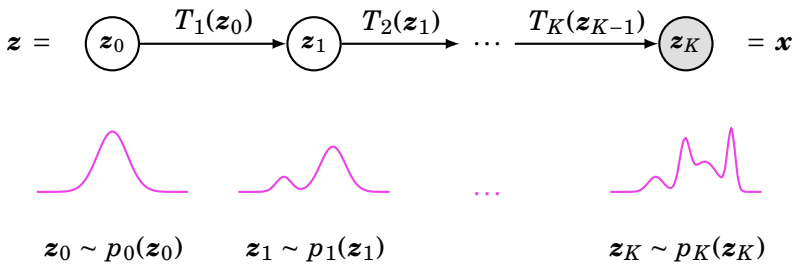


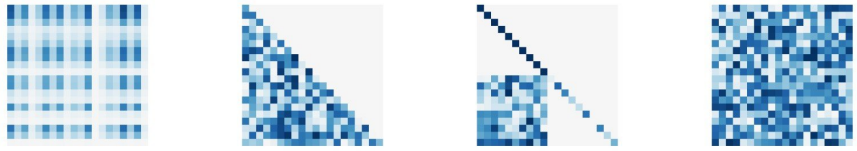
Figure 2.7: A NF is commonly composed of K finite transformations. (An alternative strategy is to construct flows in continuous time by parameterizing the flow's infinitesimal dynamics with an ordinary differential equation, e.g. in Grathwohl et al. [2018].)

The i-th dimension of the output $\boldsymbol{z}'$ is a function of the i-th dimension of the input $\boldsymbol{z}$, and is parameterized by a conditioner that accepts only variables with dimensions less than i. Because of this autoregressive property, the Jacobian of the transformation is lower triangular (Figure 2.8) and thus tractable, since the absolute-determinant of $J_{f_\phi}(\boldsymbol{z})$ can be calculated in $\mathcal{O}(D)$ time, as it is simply the product of terms on the diagonal.

$$J_{f_\phi}(\boldsymbol{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial z_1}(z_1; \boldsymbol{h}_1) & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{L}(\boldsymbol{z}) & & \frac{\partial \tau}{\partial z_D}(z_D; \boldsymbol{h}_D) \end{bmatrix}. \tag{2.19}$$

The inverse transformation f_ϕ^{-1} is simply $z_i = \tau^{-1}(z_i'; c_i(\boldsymbol{z}_{<i}))$.

Another common transformation in normalizing flows is **coupling layers**, which was first used in non-linear independent components estimation (NICE)



(a) Det. Identities (Low Rank) (b) Autoregressive (Lower Triangular) (c) Coupling (Structured Sparsity) (d) **Unbiased Est.** (Free-form)

Pathways to designing scalable normalizing flows and their enforced Jacobian structure. Residual Flows fall under unbiased estimation with free-form Jacobian.

Figure 2.8: Jacobian structures for common NF designs, image taken from Chen et al. [2019].

[Dinh et al., 2014], and also used by real non-volume preserving (Real NVP) flow [Dinh et al., 2017] and Glow [Kingma and Dhariwal, 2018]. In particular, they use **affine** coupling layers, where the transformer is an affine function $\tau(z_i; \alpha_i, \beta_i) = \alpha_i z_i + \beta_i$. Coupling layers can be thought of as a coarser version of the autoregressive flow transformation. While the fully autoregressive flow splits the input into $K = D$ parts and transforms each part as a function of all previous parts, a coupling layer splits the input into $K = 2$ parts and transforms the second part elementwise as a function of the first part [Papamakarios et al., 2019]. Concretely, the dimensions of $\boldsymbol{z}$, the input to f_ϕ, are split into two at the index $d < D$, i.e. $\boldsymbol{z} = [\boldsymbol{z}_{1:d}; \boldsymbol{z}_{d+1:D}]$. Commonly, for dimensions $1 : d$, the transformers $\tau_{1:d}$ are the identity function and the conditioners $c_{1:d}$ are constant functions such as the zero function:

$$\begin{cases} \boldsymbol{z}'_{1:d} & = \boldsymbol{z}_{1:d} \\ \boldsymbol{z}'_{d:D} & = \tau(\boldsymbol{z}_{d:D}; c(\boldsymbol{z}_{1:d})) \end{cases} \Leftrightarrow \begin{cases} \boldsymbol{z}'_{1:d} & = \boldsymbol{z}'_{1:d} \\ \boldsymbol{z}'_{d:D} & = \tau^{-1}(\boldsymbol{z}'_{d:D}; c(\boldsymbol{z}_{1:d})) \end{cases} \tag{2.20}$$

The Jacobian of coupling layers is also triangular, specifically a structured sparsity (see Figure 2.8). Again, the determinant is simply the product of the diagonal elements of the Jacobian, which are equal to the derivatives of the transformers $\tau(\boldsymbol{z}_{d:D}; c(\boldsymbol{z}_{1:d}))$.

Normalizing flows have two main usages:

- Sampling, $\boldsymbol{x} = T(\boldsymbol{z})$: Requires sampling from base distribution $p_z(\boldsymbol{z})$.
- Density estimation of $p_{\boldsymbol{x}}(\boldsymbol{x})$ with Equation 2.16: Requires computing the inverse transformation T^{-1} and its Jacobian determinant, and evaluating the density $p_z(\boldsymbol{z})$.

The desired application determines whether sampling and/or density evaluation need to be implemented, and consequently how efficient they need to be. When using affine transformers, both T and T^{-1} computationally symmetric, i.e. equally fast to evaluate or invert,

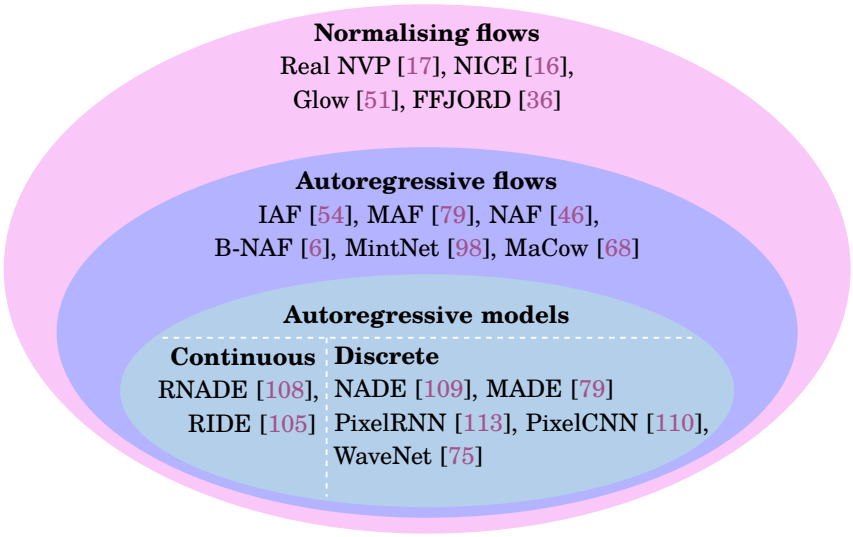


Figure 2.9: How NFs, AR flows, and AR models are related. AR flows are special cases of normalizing flows where the transformation layer is autoregressive. AR models can be shown to be single-layer flows [Papamakarios et al., 2017; Nielsen and Winther, 2020].

Autoregressive models as normalizing flows

AR models present another way to tractably model $p(\boldsymbol{x})$ is to break the model into a series of conditional distributions: $p(\boldsymbol{x}) = \prod_{i=1}^{D} p(x_i|x_1,\ldots,x_{i-1}) = \prod_{i=1}^{D} p(x_i|x_{1:i-1})$. This is the approach used, for example, by recurrent neural networks. Like normalizing flows, AR models also yield an exact density allowing for straightforward maximum likelihood, but lack latent variables.

NFs flows as described in Section 2.8 use an invertible and differentiable transformation to construct a distribution over continuous variables. Autoregressive flows are a subset of NFs, where the *conditioner* has an autoregressive constraint.

All autoregressive models of both continuous variables [Papamakarios et al., 2019] and discrete variables [Nielsen and Winther, 2020] are in fact autoregressive flows with a single autoregressive layer.

The relationship between normalizing flows and autoregressive models is illustrated in Figure 2.9. With this perspective, we can explore obtaining new autoregressive flows by stacking multiple AR models, in the same way that the Masked Autoregressive Flow (MAF, Papamakarios et al. [2017]) is obtained from stacking multiple Masked Autoencoder Distribution Estimator (MADE, Germain et al. [2015]) models.

2.9 Meta-learning

The rapidly evolving field of meta-learning, or *learning to learn*, has been defined in various inconsistent ways, even within the contemporary neural network literature. Hospedales et al. [2020] surveys decades of meta-learning history in cognitive science and machine learning, and positions recent work in contemporary deep learning-based meta-learning in a high-level problem formulation, which we will use here.

Conventional machine learning, as described in Section 2.1, is about finding the best predictive model $\hat{f}$ with parameters $\hat{\theta}$ by optimizing a **single loss function** $\mathcal{L}$ on a **single dataset** $\mathcal{D} = (x, y)_{i=1}^{n}$, where $y_i = f_{\theta,\omega}(x_i)$:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}_{\theta}(\mathcal{D}; \theta, \omega). \tag{2.21}$$

Here we make explicit the implicit specification ω, which is conventionally assumed to be fixed.

As outlined in Table 2.3, for NNs (Section 2.2), θ are the network weights, while ω encapsulates the neural architecture design, choice of optimizer, and so on. For GPR (Section 2.3), our optimized parameters θ are the kernel hyperparameters, and the loss function is the negative LML (Equation 2.13). An example of ω would be the choice of the kernel function family, which is conventionally specified *a priori*.

Meta-learning considers a **dataset of datasets** $\{\mathcal{D}_t\}_{t=1}^{T}$ drawn from an underlying **distribution of tasks** $p(\mathcal{T})$, where each task is loosely defined as a dataset and loss function $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$. The meta-level objective is to optimize a learnable ω over $p(\mathcal{T})$, i.e.

$$\omega^* = \min_{\omega} \mathop{\mathbb{E}}_{\mathcal{D},\mathcal{L}\sim p(\mathcal{T})} \mathcal{L}(\mathcal{D};\, \omega). \tag{2.22}$$

As such, we can think of ω as the meta-knowledge shared across all tasks, that specifies *how to learn* a different θ for each task.

In practice, meta-learning is generally split into two phases: **meta-training**, and **meta-testing** or online adaptation. During **meta-training**, we assume access to *source tasks* sampled from $p(\mathcal{T})$, denoted as $\mathcal{D}_{\text{source}}$. Each source dataset is split into training and validation sets, often also called the "support" and "query" sets. The purpose of meta-training is to extract a good meta-knowledge:

$$\mathcal{D}_{\text{source}} = \left\{ \left(\mathcal{D}_{\text{source}}^{\text{train}}, \mathcal{D}_{\text{source}}^{\text{val}} \right)^{(i)} \right\}_{i=1}^{M}, \qquad \omega^* = \arg\max_{\omega} \log p\left(\omega \,|\, \mathcal{D}_{\text{source}}\right). \tag{2.23}$$

Meta-testing is done on *target tasks* $\mathcal{D}_{\text{target}}$, where each target dataset is split into training and testing sets. For each target task, the base model is trained

Conventional ML: fixed ω and learnable θ		
Example	**Meta-knowledge ω**	**Task-specific θ**
NN	Hyperparameters (e.g. learning rate, optimizer, weight initialization, architecture)	NN weights
GP	Choice of kernel function family	Kernel function hyperparameters
Meta-learning: learnable ω and learnable θ		
Example	**Meta-knowledge ω**	**Task-specific θ**
Gradient-based NN hyperparameter optimization [25]	Hyperparameters (e.g. learning rate, optimizer)	NN weights
Neural architecture search (NAS) [44; 21]	Architecture	NN weights
MAML [22]	NN weights (initialization learnt from $\mathcal{D}_{\text{source}}$)	NN weights (tuned on $\mathcal{D}^{\text{train}}_{\text{target}}$)
NP [28]	NN weights	Aggregated context representation
GP with deep mean/kernel for meta-learning [Fortuin et al., 2019]	Deep mean/kernel function parameters	None (a GP, which is nonparametric, is fit on $\mathcal{D}^{\text{train}}_{\text{target}}$)

Table 2.3: **Conventional ML** versus **meta-learning**, in terms of task-general meta-knowledge ω and task-specific parameters θ.

by leveraging the meta-knowledge previously extracted from the source tasks:

$$\mathcal{D}_{\text{target}} = \left\{\left(\mathcal{D}^{\text{train}}_{\text{target}}\mathcal{D}^{\text{test}}_{\text{target}}\right)^{(i)}\right\}_{i=1}^{Q}, \qquad \theta^{*(i)} = \arg\max_{\theta} \log p\left(\theta \,|\, \omega^*, \mathcal{D}^{\text{train}}_{\text{target}}(i)\right). \tag{2.24}$$

Evaluation is then done on held-out test set $\mathcal{D}^{\text{test}}_{\text{target}}$.

Related fields

A common source of confusion is that there are other methodologies that also seek improving data efficiency by leveraging data from related tasks (Figure 2.4), such as transfer learning (TL), domain adaptation (DA), and multi-task learning (MTL). The crucial difference is that conventionally, TL, DA, and MTL use single-level optimization, whereas the salient characteristic of meta-learning is the bilevel optimization of a meta-objective [Hospedales et al., 2020]. As such, meta-learning methods can be, and have been, used to help TL, DA, MTL [Hospedales et al., 2020].

	(Meta-)training	**(Meta-)testing**
Transfer learning (TL)	$\mathcal{T}_1$	$\mathcal{T}_2$
Multi-task learning (MTL)	$\mathcal{T}_1 \cdots \mathcal{T}_N$	$\mathcal{T}_1 \cdots \mathcal{T}_N$
Meta-learning	$\mathcal{T}_1 \cdots\cdots \mathcal{T}_N$	$\mathcal{T}_{N+1} \cdots\cdots \mathcal{T}_{N+M}$

Table 2.4: Comparison of related approaches to meta-learning, adapted from Ying et al. [2018]. Each task $\mathcal{T}_i$ of size n_i, where $i \in \{1, 2, \ldots, N\}$, is either seen or unseen during training. In the literature, TL and MTL typically have large task-specific dataset sizes n_i, whereas meta-learning typically has small n_i. Furthermore, MTL typically has a small number of tasks N, while many meta-learning approaches (e.g. Garnelo et al. [2018b]; Finn et al. [2017]) rely on a larger N [Teh, 2019]. However, the field is rapidly evolving and as such, these definitions are fluid and challenged in recent work, such as Rothfuss et al. [2020] which applies meta-learning to small N.

Transfer learning (TL) [Pan and Yang, 2009] aims to leverage knowledge from one source task $\mathcal{T}_1$ to improve the learning performance required in another target task $\mathcal{T}_2$.

Multi-task learning (MTL) [Caruana, 1997; Argyriou et al., 2007] is similar to multi-task meta-learning, in that knowledge shared among multiple related tasks reinforce each other in generalization abilities. However, multi-task learning assumes that training and testing examples follow the same distribution, and does not learn to generalize to unseen tasks. Furthemore, meta-learning can sometimes be on a single task, as is often the case in online meta-RL [Hospedales et al., 2020].

Applications of meta-learning

Broadly speaking, meta-learning is a versatile framework that is particularly suited for scenarios with a collection of related datasets. Some high-level scenarios are given in the introduction chapter (see Table 1.1). In the case of image inpainting, each task corresponds to a an image, and the prior of interest could be a distribution over functions, i.e. a stochastic process. Image data can be interpreted as being generated from a stochastic process (since there are dependencies between pixel values), where each image corresponds to one realisation of the process sampled on a fixed 2-dimensional grid [Kim et al., 2019].

Other popular applications of meta-learning also include gradient-based neural network (NN) hyperparameter optimization [Franceschi et al., 2018] and

neural architecture search (NAS) [Hospedales et al., 2020; Elsken et al., 2019] (see Table 2.3).

Additionally, the meta-learning approach is often applied to the **few-shot learning (FSL)** problem [Wang et al., 2019], where a dataset is limited to only a few examples, or "shots". A common FSL benchmark is N-way K-shot classification, where $\mathcal{D}^{\text{train}}$ contains N classes, each with K "shots" (as described in e.g. Vinyals et al. [2016]). Another benchmark is few-shot function regression, which is later described in Chapter 3. Note that there are many other approaches to the general problem of FSL, of which meta-learning is just one.

2.10 Bayesian meta-learning

So far we have described meta-knowledge ω as shared parameters that work well across different tasks. Another way of looking at ω is as a means for establishing an inductive bias for learning new tasks based on old tasks, such that **learning ω is akin to "learning a prior"**. Indeed, there is a loose analogy between any meta-learning approach and hierarchical Bayesian inference [Griffiths et al., 2019]. Bayesian inference generically indicates how a learner should combine data with a prior distribution over hypotheses. Hierarchical Bayesian models (HBMs) learn that prior through experience.

In fact, the influential meta-learning algorithm Model-Agnostic Meta-Learning (MAML, Finn et al. [2017]) has been recast as an approximate HBM. Grant et al. [2018] shows that the few steps of gradient descent taken by the task-specific learners result in an approximation to the Bayesian estimate of θ for that task, with a prior that depends on the initial parameterization ω^*.

As Griffiths et al. [2019] points out, this opens opportunities to translate cognitive science insights, which has focused on HBMs, to machine learning (ML). It also invites the use of probabilistic generative models from the Bayesian deep learning toolbox for meta-learning, an example of which is Neural Processes (NPs, Garnelo et al. [2018b]). Bayesian meta-learning also comes with the added advantage of uncertainty quantification, which is vital for safety-critical FSL (e.g. medical imaging), active learning, and data-efficient exploration in reinforcement learning (RL).

Chapter 3

Methods

In the previous chapter 2, we discussed how many successful applications of deep neural networks are limited to supervised learning tasks where large volumes of labeled instances are available. Gaussian processes (GPs) are a counterpoint to NNs in terms of their data-efficiency and hierarchical Bayes flexibility (properties 3 and 4 in Figure 2.1 respectively), thus hinting at avenues for combining their advantages, which are the subject of this chapter.

This chapter introduces several key existing deep generative models that lie on the spectrum between GPs and NNs, and explain how they can be combined in different ways. In particular, we focus on NPs, which are a neural network implementation of stochastic processes.

We also previously discussed in Section 2.3 how GPs provide an alternative function space approach to machine learning, directly placing a distribution over functions. We then discuss how GPs (and stochastic processes in general) can be seen as methods for meta-learning, in particular few-shot function estimation.

3.1 Combining Gaussian processes and neural networks

There have been many deep generative models that seek to combine GPs and NNs, some of which we highlight in Figure 3.1. The setting for the selected work in Figure 3.1 is to, much like GPs, given a set of input-output pairs $\boldsymbol{x}_n \in \mathbb{R}^{d_x}, \boldsymbol{y}_n \in \mathbb{R}^{d_y}$, learn a **distribution over functions** $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ that model nonlinear relationships between pairs of random variables $\boldsymbol{x}$ and $\boldsymbol{y}$. This setting can be referred to as meta-learning.

3.2 Neural Processes

Neural Processes (NPs, Garnelo et al. [2018b]) are a class of neural network-based models that approximate stochastic processes. This gives NPs the benefits of GPs such as flexibility and uncertainty estimates, while retaining the benefits of NNs

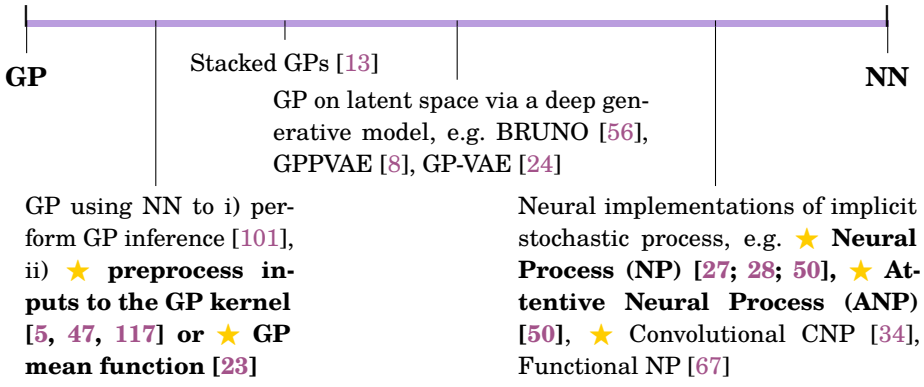


Figure 3.1: Selected work on the spectrum between NNs and GPs. Work labeled with ★ will be explored in-depth in Methods (this chapter) and Experiments (Chapter 3).

such as powerful and flexible function approximation, and scalable training and inference on large datasets.

Since NPs are mimicking stochastic processes, they can be described as meta-learning algorithms for the setting where a large collection of small-but-related datasets are available, as we have discussed in Section 2.10. Following the meta-learning problem formulated in Section 2.9, NPs are trained on a collection of datasets where each dataset of tuples $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^n$ is interpreted as an instantiation of a function of interest $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ drawn from the underlying data-generating stochastic process G, where $y_i = f_j(x_i)$ for $i = 1, \ldots, n$.

Within each dataset, we select a subset as the context set $\{\boldsymbol{X}_C, \boldsymbol{Y}_C\} \coloneqq (\boldsymbol{x}_c, \boldsymbol{y}_c)_{c \in C}$, where $C \coloneqq \{1, \ldots, m\}$ refers to the indices of the context set of size $m \leq n$. Another subset called the target set $\{\boldsymbol{X}_T, \boldsymbol{Y}_T\} \coloneqq (\boldsymbol{x}_t, \boldsymbol{y}_t)_{t \in T}$ is also selected, and in common practice, the entire dataset of n data pairs is used as the target set where $T = \{1, \ldots, n\}$ such that $C \subset T$, although the NP is defined for any arbitrary C and T. We are interested in predicting the target output $\boldsymbol{y}_T$ given the target input as well as the context set, $p(\boldsymbol{Y}_T \mid \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{Y}_C)$. In Gaussian processes, this posterior predictive distribution has an analytic form as shown previously in Equation 2.7, whereas in Neural Processes, this distribution is modeled with neural networks. Stochastic processes can generally be described via a consistent family of conditional distributions; much like GPs, NPs can be described via its posterior predictive distributions. NPs approximate an (infinite) family of conditional distributions $p(\boldsymbol{Y}_T \mid \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{y}_C)$ that are invariant to ordering of the contexts and ordering of the targets. NP as approximating the conditionals of the consistent data-generating stochastic process

Many NP variants have been proposed that differ in either the generative modeling (e.g. specifying latent variables) or the neural network architecture

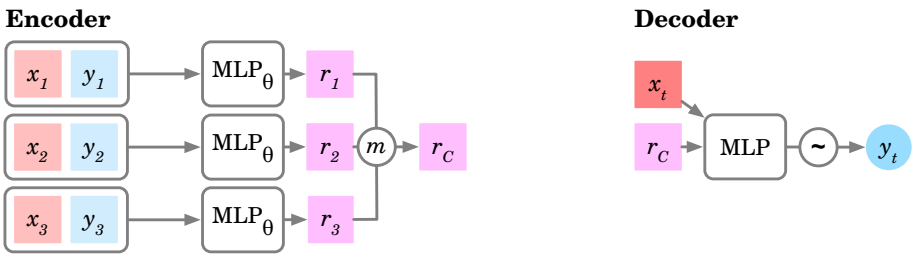


Figure 3.2: Conditional Neural Process (CNP), a NP with deterministic encoder. Square colored nodes represent deterministic variables while circles represent stochastic variables.

(e.g. attention, convolutions). These choices can have significant impact on the downstream performance [Le et al., 2018].

Conditional Neural Process (CNP)

The first proposed variant was the Conditional Neural Process [Garnelo et al., 2018a]. As illustrated in Figure 3.2, the CNP has a simple model specification where it encodes each pair $(\boldsymbol{x}_c, \boldsymbol{y}_c)$ in context set $\{\boldsymbol{X}_C, \boldsymbol{Y}_C\}$ via a multilayer perceptron (MLP) to obtain individual representations $\boldsymbol{r}_c \in \mathbb{R}^d$, which are then aggregated using a commutative operation such as the mean m to produce a global representation for all context points, $\boldsymbol{r}_C \in \mathbb{R}^d$, with permutation invariance in C. In the decoder, another MLP then accepts both the context embedding $\boldsymbol{r}_C$ and some target input $\boldsymbol{x}_t$ to produce the parameters for a Gaussian distribution $\mathcal{N}\left(\boldsymbol{y}_t \,|\, \mu_y, \sigma_y\right)$. As such, the conditional predictive distribution $p(\boldsymbol{Y}_T \,|\, \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{y}_C) = p(\boldsymbol{y}_T \,|\, \boldsymbol{x}_T, \boldsymbol{r}_C)$ is modelled as a Gaussian factorised across the target outputs $\boldsymbol{y}_T$, that is, $p(\boldsymbol{y}_T) = \prod_{t \in T} \mathcal{N}\left(\boldsymbol{y}_t \,|\, \mu_y, \text{diag}(\sigma_y^2)\right)$. For further details on the model architecture, refer to Section 4.1.

The CNP is straightforwardly trained by maximizing the log predictive probability of the true targets given the contexts, $\mathcal{L} = \mathbb{E}_{\boldsymbol{X}_C, \boldsymbol{Y}_C, \boldsymbol{X}_T, \boldsymbol{Y}_T}$.

Neural Process (NP)

One drawback of Conditional Neural Process (CNPs, Garnelo et al. [2018a]) is that they cannot produce different function samples for the same context data—the mean and variance of the target output y_T will always be the same given the same target input x_T and context points.

In order to allow for sampling of different coherent samples conditioned on context observations, Garnelo et al. [2018a] introduce a latent variable on top of the original CNP model, by letting global representation r to parameterize a Gaussian distribution over a global latent variable, $z \sim \mathcal{N}\left(\mu(r), \sigma(r) \odot I\right)$. The latent distribution z is then sampled once and passed into the decoder.

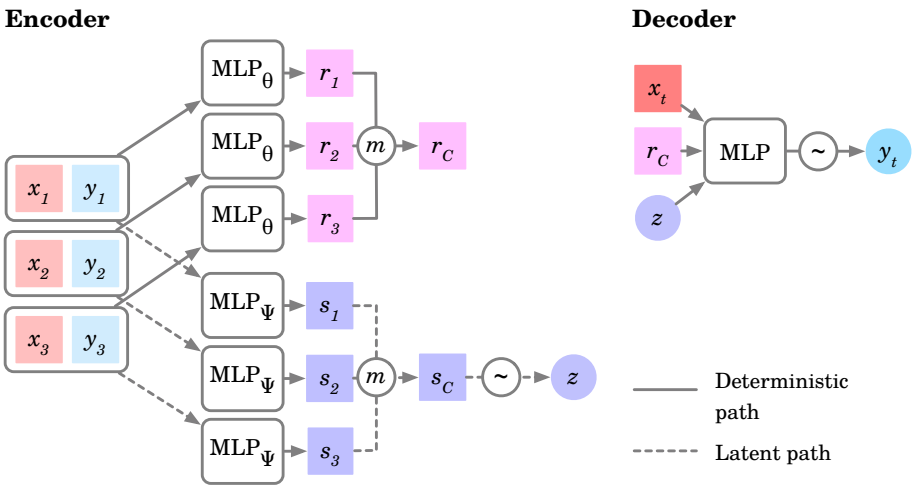


Figure 3.3: Neural Process (NP), a NP with latent variable. In Garnelo et al. [2018b] only the latent path is used, but the deterministic path is kept in Garnelo et al. [2018a] and Kim et al. [2019]. Keeping the deterministic path is shown to be beneficial quantitatively and qualitatively [Le et al., 2018].

Unlike the CNP, inference in the NP and Attentive Neural Process (ANP, Kim et al. [2019]) is less straightforward. The parameters of the encoder and decoder are learnt by optimising the evidence lower-bound (ELBO) to the log predictive likelihood, which in itself is intractable:

$$\begin{aligned}&\log p(\boldsymbol{Y}_T \mid \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{Y}_C)\\ \geq &\mathbb{E}_{q(\boldsymbol{z}\mid \boldsymbol{s}_T)}[\log p(\boldsymbol{Y}_T \mid \boldsymbol{X}_T, \boldsymbol{r}_C, \boldsymbol{z})] - D_{\mathrm{KL}}(q(\boldsymbol{z}\mid \boldsymbol{s}_C, \boldsymbol{s}_T) \| q(\boldsymbol{z}\mid \boldsymbol{s}_C)),\end{aligned}$$

much like the VAE's variational lower bound as described in the previous chapter (Equation 2.14). The first term is the reconstruction error of target outputs from the target inputs, learnt context representation, and latent summary. The second term regularizes the first, being the KL divergence between the latent summary of the context with and without the target.

Attentive Neural Process (ANP)

Attentive Neural Processes (ANP) (Figure 3.4 are a generalization of NPs that uses attention mechanisms to learn to select relevant information among contexts and targets. Kim et al. [2019] hypothesized that the mean aggregation step assigns equal weight to all context points, resulting in the problem of underfitting found in NPs, where the decoder could not predict the context points even though they were observed inputs. By incorporating attention into the NP framework to

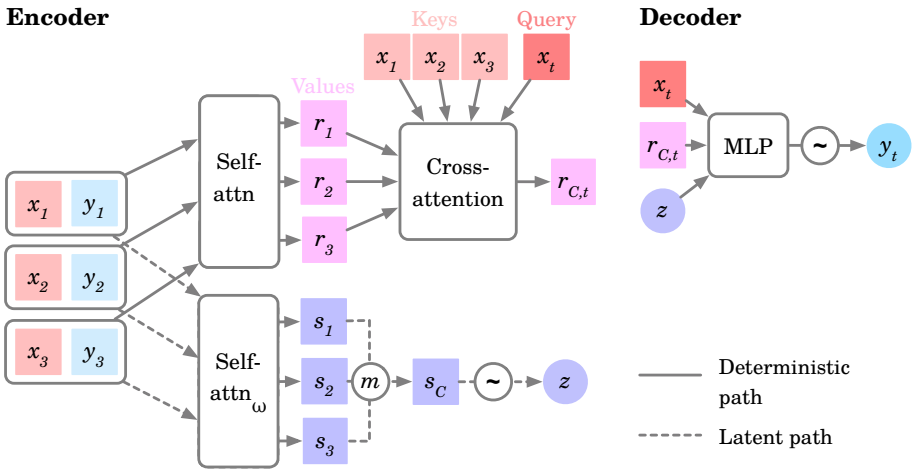


Figure 3.4: Attentive Neural Process (ANP). Square nodes represent deterministic variables while circles represent stochastic variables.

help the decoder learn which context points provide relevant information for a given target prediction, performance was improved.

A downside of attention is that the ANP no longer scales linearly. Assuming n_C context points and n_T target points, where $C \subset T$, the cross-attention in the deterministic path entails n_T queries attending to n_C key-value pairs. The complexity of this operation is $\mathcal{O}(n_C n_T)$, up fro $\mathcal{O}(n_T)$. Nevertheless, in practice, the wall time is not much higher as the attention operations are some form of matrix multiplication which can be parallelized with a GPU.

3.3 Meta-GPs

Gaussian process regression (GPR) can be straightforwardly extended to the meta-learning setting by simply sharing mean and/or kernel hyperparameters across many datasets, as formalized in Fortuin and Rätsch [2019] (Figure 3.5). In a "vanilla" GP with a constant mean function and a squared exponential (SE) kernel, the mean function simply has a single constant as its hyperparameter, and the SE has two hyperparameters: lengthscale and output variance or scale. We can also use deep NNs in the mean and/or kernel function(s), in which case the hyperparameters would be the weights of the NN.

Deep kernel learning (DKL, Wilson et al. [2016]) [Wilson et al., 2016], also known as manifold Gaussian processes [Calandra et al., 2016], refers to transforming inputs to some base kernel with a NN, such that while the base kernel can usually discover a stationary structure with an interpretable but restrictive form, the nonlinear NN transformation additionally captures nonstationary and hierarchical structure. These deep kernels can be used to replace standard kernels

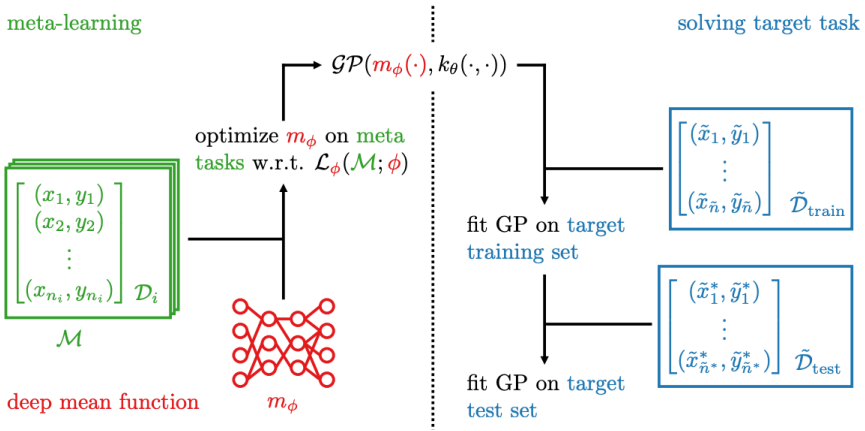


Figure 3.5: Meta-learning GP, where the mean function is given by a deep NN. More generally, the mean and kernel functions can be anything, but their hyperparameters are optimized over a set of meta-training tasks during the "meta-learning" stage, with the goal of generalizing to meta-testing target task(s). Taken from Fortuin and Rätsch [2019, Figure 1].

in methods such as GPR. In a similar vein, we can also replace the conventional zero-mean or constant-mean function in GPR with a NN.

The approach in Fortuin and Rätsch [2019] is also similar to GPShot [Patacchiola et al., 2019], which uses the same kernel for all tasks but does not include deep mean learning, and adaptive deep kernel learning [ADKL Tossou et al., 2019], which also includes an additional module for encoding each task separately.

In short, what Fortuin and Rätsch [2019] formalizes as meta-learning with GPs places GPR in the training regime of NPs, making them directly comparable. Conventional ML with GPR is usually trained on observations from a single function, while NPs are trained on multiple functions from a stochastic process.

Chapter 4

Experiments

We compare four methods from the NP family (Section 3.2), and four methods from the class of meta-learning GPs (Section 3.3),

1. Neural Process (NP)
2. Conditional Neural Process (CNP)
3. Attentive Neural Process (ANP)
4. Attentive Conditional Neural Process (ACNP)
5. GP with a constant mean and SE kernel
6. GP with a constant mean and NN kernel
7. GP with an NN mean and SE kernel
8. GP with an NN mean and NN kernel

For the NP models, we mainly relied on conventions set in Le et al. [2018], which does an exhaustive comparison of various objectives and model specifications. For the meta-GP models, we used the setup from Rothfuss et al. [2020], which also included a comparison the to the NP model but not the attentive and conditional variants.

4.1 Model architectures

For the following architecture descriptions, we outline the shapes of the tensors (different dimensions are separated by a $\times$) at each consecutive transformation (marked by an annotated arrow). In practice, each tensor is prepended by a batch dimension, $B\times$, for which we use batch size $B = 16$.

For the deterministic path used in all four NP models, the **deterministic (attentive) encoder** $r(\boldsymbol{x}_C, \boldsymbol{y}_C[, \boldsymbol{x}_t])$ produces a deterministic representation $\boldsymbol{r}_C$ from context set $\{\boldsymbol{x}_C, \boldsymbol{y}_C\} \coloneqq (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i \in C}$, where $C = \{1, \dots, m\}$, and optionally also accepts target input $\boldsymbol{x}_t$ for some t if cross-attention is used:

$$|C| \times \overbrace{(d_x + d_y)}^{\substack{\text{concatenated } (x_i, y_i) \\ \text{for each } i \in C}} \xrightarrow{\text{lin+relu}} \underbrace{|C| \times \overbrace{128}^{\substack{r_i \text{ at the} \\ \text{final layer}}}}_{\text{3 times (3-layer MLP)}} \xrightarrow{\text{mean}} \overbrace{128}^{r_C}$$

To include cross-attention, instead of taking the mean across all $(r_i)_{i \in C}$ in the last step, we additionally take in (2-layer MLP-transformed) context inputs $\boldsymbol{x_C} \coloneqq (\boldsymbol{x_c})_c$ and the target input $\boldsymbol{x}_t$, and performs multihead attention. Each target query $\boldsymbol{x}_t$ attends to the context inputs to produce a query-specific representation r_*. To include self-attention, instead of the 3-layer MLP, stack 2 layers of self-attention to produce $|C|$ representations r_i for $i \in C$. Self-attention uses the same architecture as cross-attention but with identical keys and queries $k_i = v_i$, and $q = k_j$ for each $j \in C$.

Latent encoder $q(z|x_C, y_C)$ to produce latent representation z:

$$|C| \times (d_x + d_y) \underbrace{\xrightarrow{\text{lin+relu}} |C| \times 128}_{\text{3 times (3-layer MLP)}} \xrightarrow{\text{mean}} \overbrace{128}^{\boldsymbol{s}_C} \xrightarrow{\text{lin+relu}} 2 * 128 \xrightarrow{\text{split}} \begin{cases} \overbrace{128}^{\boldsymbol{\mu}_z} \\ \underbrace{128}_{\boldsymbol{\sigma}_z} \end{cases}$$

The latent path outputs $\boldsymbol{\mu}_z, \boldsymbol{\sigma}'_z \in \mathbb{R}^d$ where $\boldsymbol{\sigma}_z = 0.1 + 0.9\sigma(\boldsymbol{\sigma}'_z)$ and σ is the sigmoid function. These parameterise $q(\boldsymbol{z}|\boldsymbol{s}_C) = \mathcal{N}\left(\boldsymbol{z}|\boldsymbol{\mu}_z, 0.1 + 0.9\boldsymbol{\sigma}(\sigma z)\right)$.

Similarly, the **decoder** $f_{z,r_C}(x_T)$ outputs $\boldsymbol{\mu}_y, \boldsymbol{\sigma}'_y \in \mathbb{R}^d$ where $\boldsymbol{\sigma}_y = 0.1 + 0.9f(\boldsymbol{\sigma}'_y)$ and f is the softplus function. These parameterize a Gaussian factorised across target outputs y_T, $p(\boldsymbol{y}_i|\boldsymbol{z}, \boldsymbol{X}_C, \boldsymbol{Y}_C, \boldsymbol{x}_i) = \mathcal{N}\left(\boldsymbol{y}_i|\boldsymbol{\mu}_y, \text{diag}(\boldsymbol{\sigma}_y)\right)$.

$$|T| \times \overbrace{(d_x + d_z + d_r)}^{\text{concatenated } (x_i, y_i) \text{ for each } i \in T} \underbrace{\xrightarrow{\text{lin+relu}} |T| \times 128}_{\text{3 times (3-layer MLP)}} \xrightarrow{\text{lin+relu}} 2 * d_y \xrightarrow{\text{split}} \begin{cases} \overbrace{d_y}^{\boldsymbol{\mu}_y} \\ \underbrace{d_y}_{\boldsymbol{\sigma}_y} \end{cases}$$

For the ANP, the last step takes in x_i and performs multihead attention. We use the same decoder architecture for all experiments, and 8 heads for multihead, like in [Kim et al., 2019].

For the meta-GP models, the base kernel is always an SE kernel with an optional NN mapping. The mean function is either a learnt constant function, or directly parameterized by an NN. The NN mean and kernel modules follow the same architecture as the MLP blocks in the NP decoders, with an additional output layer. The output layer size for the kernel NN is the number of input features, while the output layer size for the mean NN is 1 (the output dimensionality).

4.2 Datasets

While majority of previous work on meta-learning, including NPs, assumes that tasks $\mathcal{T} \sim p(\mathcal{T})$ are of large or infinite supply during meta-training, we use limited data sizes which are more realistic, following the experiments from Rothfuss et al. [2020].

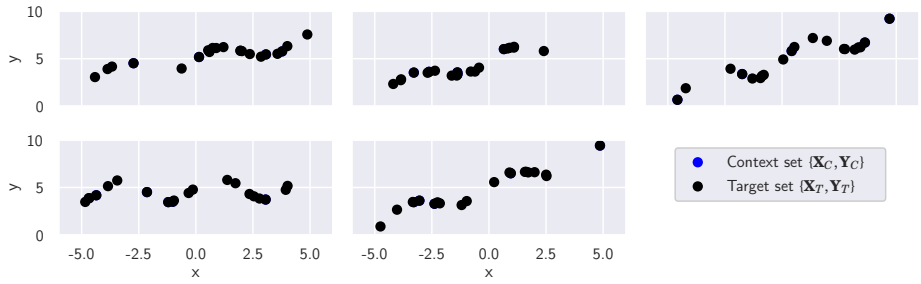


Figure 4.1: 6 sample sinusoidal tasks from the full 200 meta-testing tasks. For each curve, 5 points are randomly chosen as the context set, while the full 20 points are the target set. Only 20 tasks with 20 samples each were used for meta-training.

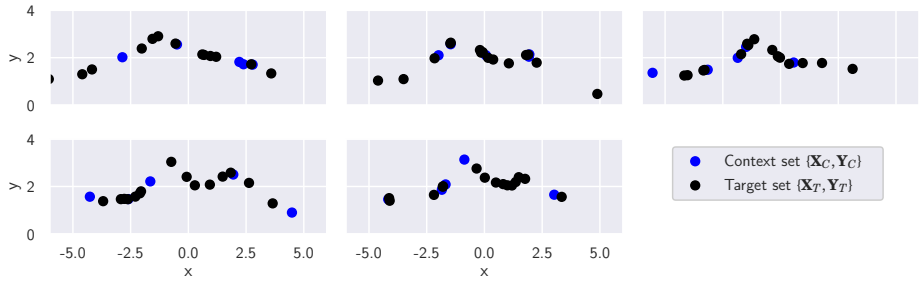


Figure 4.2: 6 sample Cauchy tasks from the full 200 meta-testing tasks. The dataset sizes are identical to the sinusoidal tasks.

4.2.1 Synthetic

We use two regression datasets. The first consists of sinusoid functions with a range of amplitudes, phase-shifts, slope and intercept (Figure 4.1). The second is the density of 1-dimensional mixtures of Cauchy distributions plus random functions sampled from a GP prior with an SE kernel (Figure 4.2). Further details can be found in Rothfuss et al. [2020, Appendix C].

4.2.2 Physionet

The Physionet 2012 challenge dataset consists of 12,000 ICU stay records. Of the 4,000 records in set A, we sample 200 patients for meta-training and 200 patients for meta-testing. Typically, the dataset is used for mortality prediction from 37 variables. However, following Rothfuss et al. [2020], we select only the GCS (Glasgow Coma Score) variable and perform 1D regression by taking the first 24 hours as context and predicting GCS over the full 48 hours (Figure 4.3).

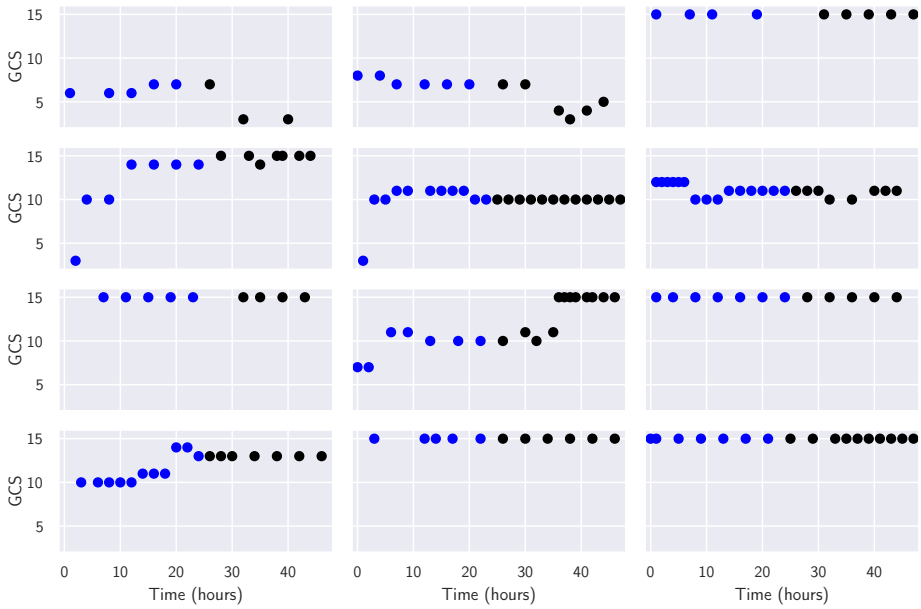


Figure 4.3: The Physionet-GCS dataset consists of GCS measurements as an integer from 3–15, taken over 48 hours. Data from the first 24 hours (4–24 points) are taken as the context set.

4.3 Training and evaluation

Overall, evaluating a meta-learner consists of two phases, meta-training and meta-testing. The main differences lie in the meta-training objectives. While the GP models maximize the marginal likelihood directly, the CNP models minimize a marginal likelihood conditioned on context points, while the latent NP models minimize the evidence lower bound (ELBO), a variational lower bound for the conditional marginal likelihood (or "evidence").

Meta-training

The meta-learner is provided with a set of M_{train} tasks and optimizes its respective meta-objective on each batch of tasks at every iteration, in order to learn the parameters of the neural networks or mean/kernel function parameters. The synthetic datasets use $M_{\text{train}} = 20$ while the Physionet dataset uses $M_{\text{train}} = 100$.

NPs minimize the ELBO to the log predictive likelihood: $\log p(\boldsymbol{Y}_T \mid \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{Y}_C)$ either directly (Section 3.2) or approximately (Section 3.2). If there are n samples in the task, the n_C context points can be between 10% and 90% of the total points, that is, $n_C \sim U(0.1n, 0.9n)$. During meta-training, NPs randomize context sizes to encourage the learned model to be able to handle different sizes and positions of the context set during meta-testing [Garnelo et al., 2018a,b].

The GP models do not select a context set but use the entire target set and maximize its log marginal likelihood $\log p(\boldsymbol{Y}_T | \boldsymbol{X}_T)$ (Section 2.3).

We train each model's parameters with Adam, an extension of stochastic gradient descent (SGD), for 5000 iterations; any longer and some of the models are prone to overtraining, which we describe in Section 5.3.

Meta-testing

Once the parameters of the GP prior or the NP model are optimized on the meta-training tasks, we proceed to the meta-testing stage, which is almost identical for all the models and datasets. Each of the $M_{\text{test}} = 200$ tasks is split into a context and target set, with the number of context points being 5 for the synthetic datasets, and 4–24 for the Physionet dataset. The NPs do a single forward pass through the encoder-decoder networks to obtain $\log p(\tilde{\boldsymbol{Y}}_T \mid \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{Y}_C)$ as a factorized Gaussian. The GPs, as shown on the right side of Figure 3.5, perform posterior inference by first fitting the GP to $\{\boldsymbol{X}_C, \boldsymbol{Y}_C\}$, and then evaluating the predictive posterior distribution of the GP on the test points $\{\boldsymbol{X}_T, \boldsymbol{Y}_T\}$ to yield $\log p(\tilde{\boldsymbol{Y}}_T \mid \boldsymbol{X}_T, \boldsymbol{X}_C, \boldsymbol{Y}_C)$, which is a closed-form expression when using an exact GP with Gaussian likelihood (Equation 2.9).

4.4 Results

We compare the predicted targets $\tilde{\boldsymbol{Y}}_T$ to the actual targets $\boldsymbol{Y}_T$ using various evaluation metrics, namely, negative log-likelihood (NLL), root-mean-square error (RMSE), and calibration error (CE).

The 8 methods are evaluated across several experimental setups, namely

- 1D regression on synthetic sinusoidal and Cauchy functions.
- 1D regression on real-world data from the Physionet Challenge 2012.

We look at NLL, a metric which captures both accuracy and uncertainty calibration. We also separately measure these aspects using RMSE and CE [Kuleshov et al., 2018], respectively.

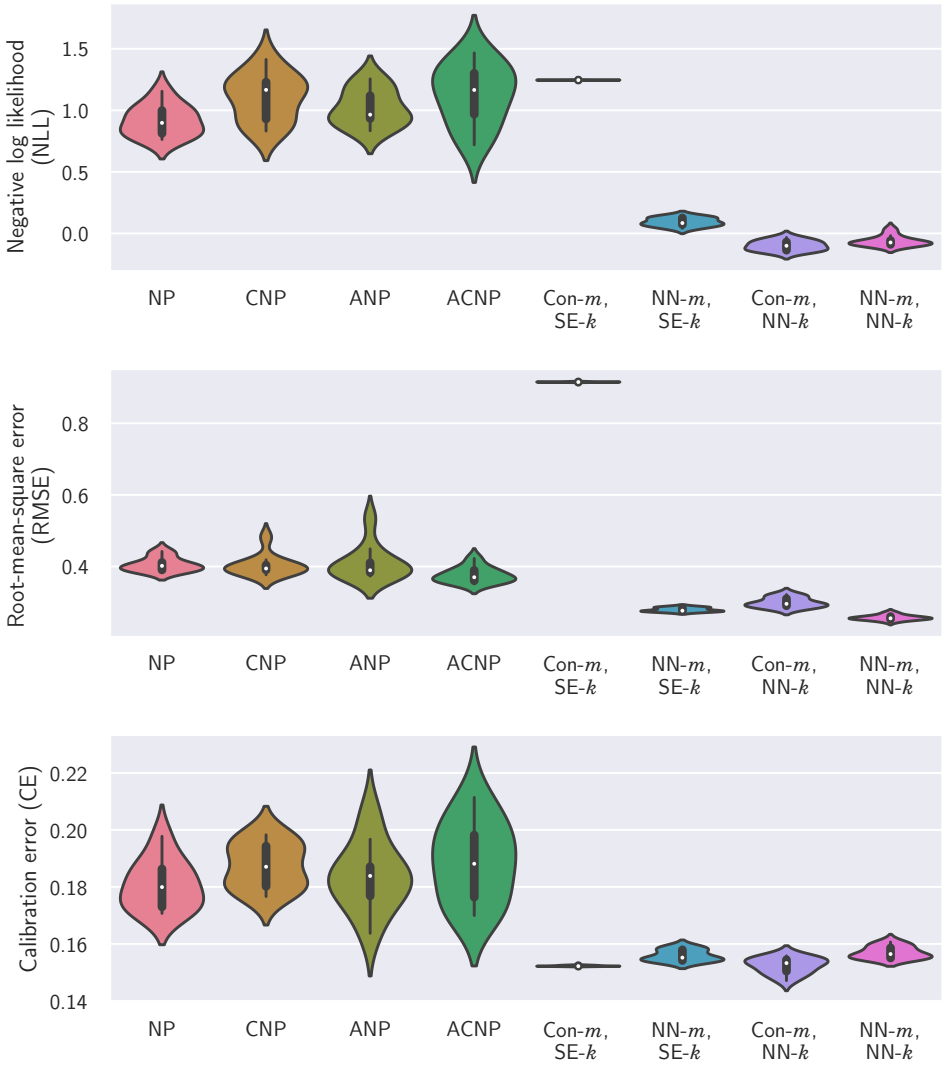


Figure 4.4: Comparison of 4 NP and 4 meta-GP methods on a dataset of **sinusoidal functions**, over 10 random seeds. Lower values are better. Actual numbers are reported in the corresponding Table 4.1.

Table 4.1: Mean and standard deviation of each method's evaluation metrics.

	NP	CNP	ANP	ACNP	Con-m, SE-k	NN-m, SE-k	Con-m, NN-k	NN-m, NN-k
NLL	0.915 ±0.125	1.105 ±0.190	1.020 ±0.147	1.125 ±0.242	1.246 ±0.000	0.092 ±0.034	**−0.09** ±0.041	−0.06 ±0.041
RMSE	0.405 ±0.019	0.403 ±0.030	0.409 ±0.049	0.376 ±0.021	0.915 ±0.000	0.279 ±0.005	0.299 ±0.013	**0.256** ±0.007
CE	0.180 ±0.008	0.187 ±0.007	0.183 ±0.011	0.188 ±0.014	**0.152** ±0.000	0.155 ±0.001	**0.152** ±0.002	0.156 ±0.002

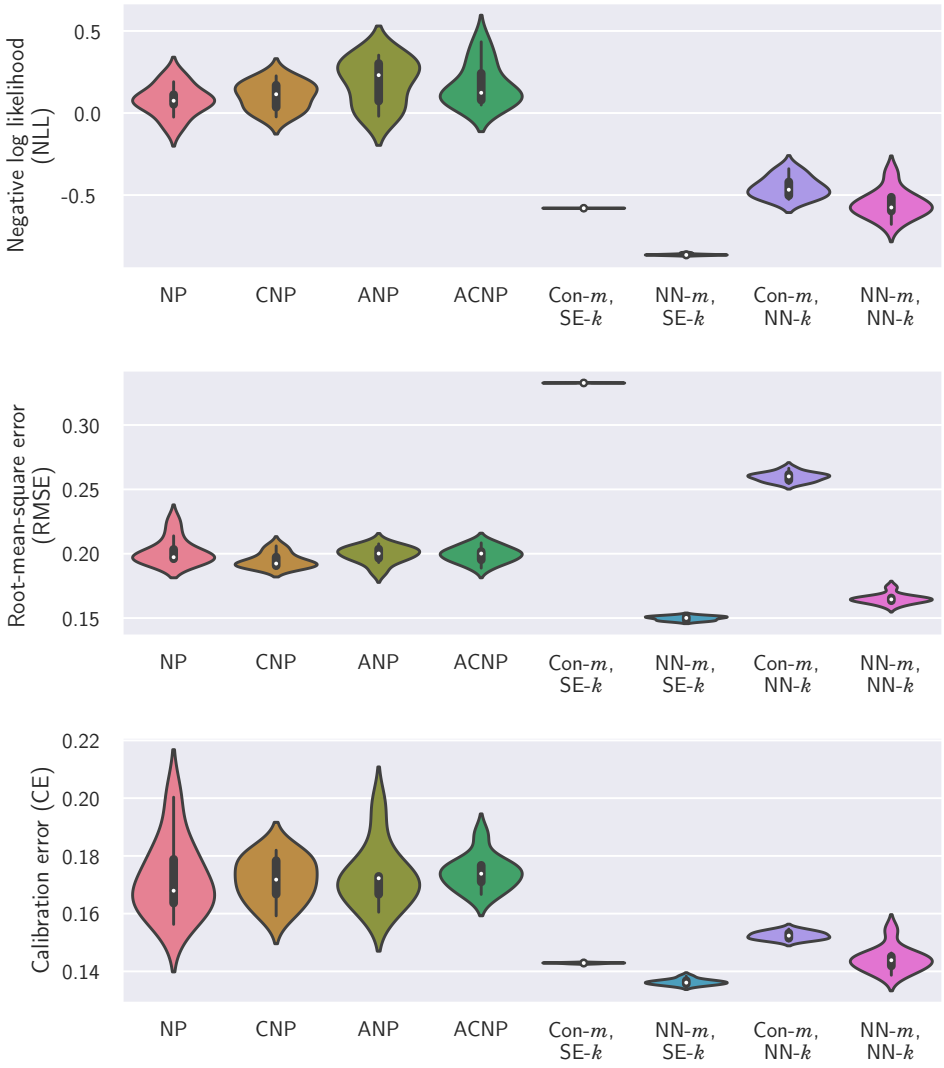


Figure 4.5: Comparison of 4 NP and 4 meta-GP methods on a dataset of **Cauchy functions**, over 10 random seeds. Lower values are better. Actual numbers are reported in the corresponding table (4.2).

Table 4.2: Mean and standard deviation of each method's evaluation metrics.

	NP	CNP	ANP	ACNP	Con-m, SE-k	NN-m, SE-k	Con-m, NN-k	NN-m, NN-k
NLL	0.076 ±0.091	0.104 ±0.083	0.187 ±0.140	0.170 ±0.128	−0.58 ±0.000	**−0.86** ±0.004	−0.45 ±0.063	−0.55 ±0.084
RMSE	0.201 ±0.010	0.194 ±0.005	0.199 ±0.006	0.199 ±0.005	0.332 ±0.000	**0.149** ±0.001	0.259 ±0.003	0.164 ±0.003
CE	0.172 ±0.013	0.171 ±0.007	0.172 ±0.010	0.174 ±0.005	0.142 ±0.000	**0.136** ±0.000	0.152 ±0.001	0.144 ±0.004

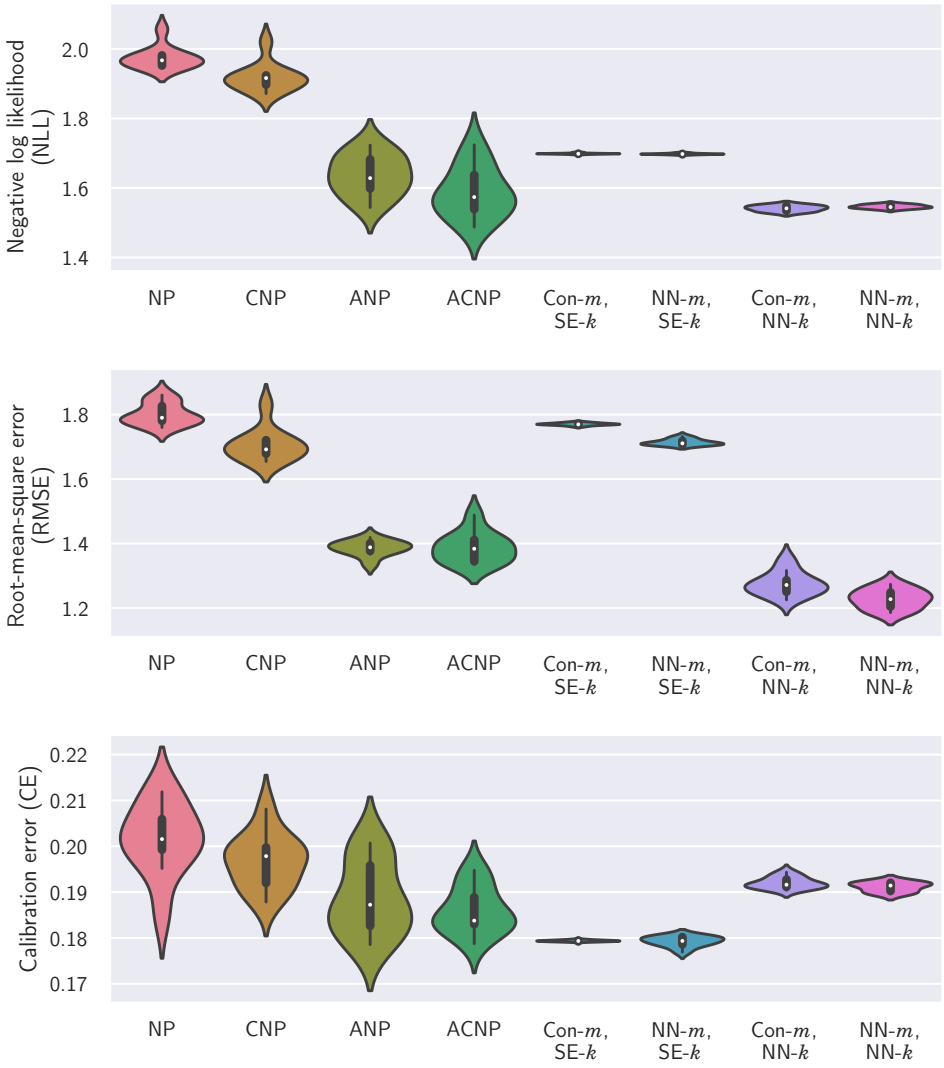


Figure 4.6: Comparison of four NP and four meta-GP methods on a dataset of **Physionet functions**, over 10 random seeds. Lower values are better. Actual numbers are reported in the corresponding table (4.3).

Table 4.3: Mean and standard deviation of each method's evaluation metrics.

	NP	CNP	ANP	ACNP	Con-m, SE-k	NN-m, SE-k	Con-m, NN-k	NN-m, NN-k
NLL	1.974 ±0.032	1.917 ±0.040	1.636 ±0.058	1.587 ±0.072	1.698 ±0.001	1.697 ±0.001	**1.540** ±0.008	1.545 ±0.005
RMSE	1.801 ±0.034	1.705 ±0.050	1.385 ±0.023	1.386 ±0.047	1.770 ±0.003	1.713 ±0.009	1.274 ±0.036	**1.227** ±0.030
CE	0.201 ±0.007	0.196 ±0.005	0.188 ±0.007	0.185 ±0.005	**0.179** ±0.000	**0.179** ±0.001	0.191 ±0.001	0.191 ±0.001

Chapter 5

Discussion

On the synthetic datasets (Figures 4.4 and 4.5), the four NP models are roughly on par with each other, and are all outperformed by most of the GP models. On the more challenging Physionet dataset (Figure 4.6), the attentive NPs are given a chance to shine and are comparable to the best GP models with an NN kernel.

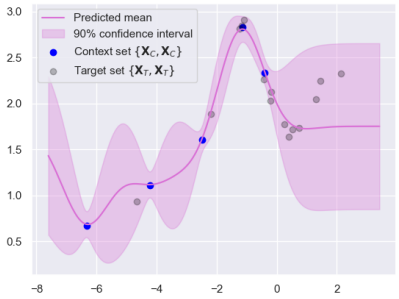
We also note that on different datasets, the best performing GP model overall (in terms of predictive log-likelihood) depends on the properties of the data. This is supported both by the statistics across all 200 meta-test tasks and across 10 random seeds, as well as regression plots of individual meta-test tasks for illustrative purposes. This gives us the freedom to specify our prior knowledge of known properties of the data as either a mean or kernel function, while the rest of the properties are learnt directly from the data with an NN mean or kernel function.

5.1 Impact of handcrafted mean and kernel functions

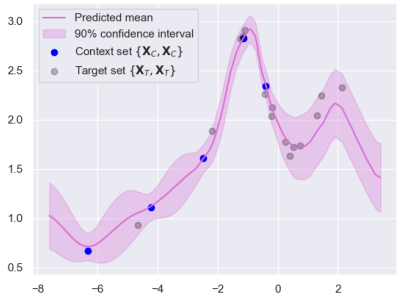
Across the datasets, the "con-m, SE-k" or "vanilla" GP has poor RMSE but good CE. The high RMSE stems from the simple model being unsuitable for the various datasets in different ways, while the low CE is from the high uncertainty of the model in areas without context points.

The constant mean is unsuitable for the Cauchy dataset, as it results in mean-reverting behaviour outside of the context domain that fails to extrapolate well. To illustrate this, on the right end of the Cauchy function shown in Figure 5.1a, the GP reverts to the constant mean and consequently misses the target points below the mean. Nevertheless, it is able to "know when it does not know" and have a high uncertainty at the tail ends, resulting in better CE compared to the NPs. The unsuitability of the constant mean is true for most samples from the Cauchy dataset (Figure 4.2), which exhibit different behaviour at the tails compared to the rest of the function. The constant mean, even with a learnt NN kernel, still has poor RMSE on the Cauchy data (Figure 4.5).

However, when we replace the constant mean with a neural network (NN) on the Cauchy dataset, the RMSE improves significantly overall. As an exam-

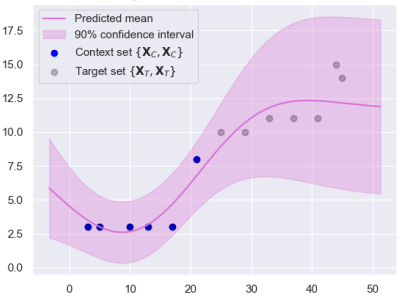


(a) "con-m, SE-k" GP on Cauchy.

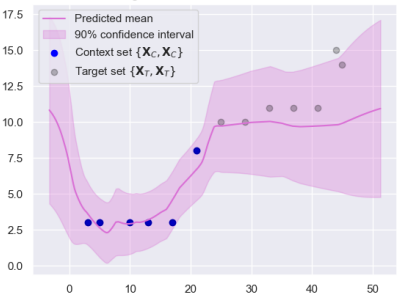


(b) "NN-m, SE-k" GP on Cauchy.

Figure 5.1: The constant mean in Figure 5.1a results in undesirable mean-reversion at the tail. With an NN mean, in Figure 5.1b, the model learns to extrapolate the hump at the tail.



(a) "Con-m, SE-k" GP on Physionet.



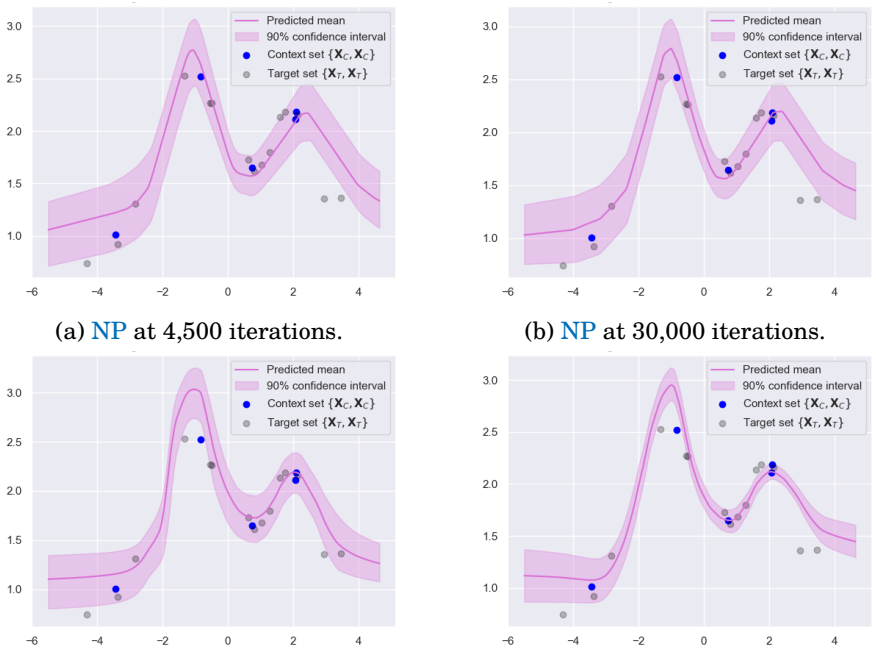
(b) "Con-m, NN-k" GP on Physionet.

Figure 5.2: By replacing the SE kernel in Figure 5.2a with an NN kernel in Figure 5.2b, the GP can respond to the step-like jump in the middle. Without it, the GP using the SE kernel smoothes out the curve in order to best fit the data.

ple, when using an NN mean with an SE kernel, the model is able to correctly extrapolate to the hump on the right (Figure 5.1b).

While the SE kernel is a good fit to the true Cauchy curves, this is not the case for the Physionet dataset. The stationary SE kernel function is not adequate if the target function is not smooth, such as the Physionet dataset which can have step-like jumps (Figure 4.3). The overly smooth "con-m, SE-k" GP misses target points, as shown in Figure 5.2a.

One way to obtain a non-stationary covariance function is by warping the inputs to a stationary covariance function through a nonlinear mapping such as a neural network (NN)—this is known as DKL (Section 3.3). The intuition is that the neural network only needs to identify its discontinuities while for the remaining part the model can rely upon the SE kernel. While constant mean is suitable for Physionet as many samples are constant functions (Figure 4.3), we can use an NN to warp the inputs to a base SE kernel so that the GP can learn rapid changes (e.g. Figure 5.2b).



(a) NP at 4,500 iterations.

(b) NP at 30,000 iterations.

(c) ANP at 4,500 iterations.

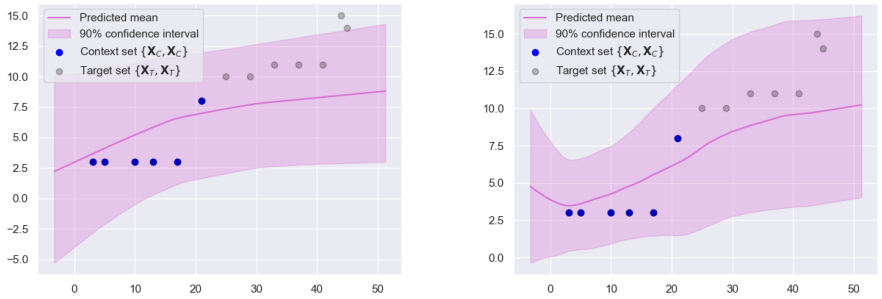
(d) ANP at 30,000 iterations.

Figure 5.3: When overtraining the NP and ANP on a meta-test Cauchy function, the uncertainty reduces only for the ANP.

5.2 Role of attention in Neural Processes

Generally, attention improves performance since it expands the range of functions that can be modelled (Section 3.2). However, uniformly weighting the context points seems to have been already sufficient on the simpler synthetic functions, so the attentive and non-attentive variants are on par (Figures 4.4 and 4.5). On the other hand, attention proved useful on the Physionet dataset, resulting in lower evaluation metrics on both accuracy and/or uncertainty calibration compared to the non-attentive counterparts (Figure 4.6). This is most likely based on two factors: the time series nature of the Physionet data, and the properties of the functions. The context set is always the first 24 hours (Figure 4.3) so it may be natural to say, attend to the recent data points more than the older data points, instead of uniform attention. Furthermore, the Physionet functions often seem to have almost discontinuous sections separated by step-like jumps, where it may be more sensible to attend only to context points from the relevant section as the target point.

However, the increase in model flexibility also comes at the risk of overconfidence if trained for too long. As seen in Figure 5.3, the ANP becomes significantly more overconfident as training progresses, with the 90% confidence band becom-



(a) NP on Physionet.

(b) ANP on Physionet.

Figure 5.4: Both the NP and attentive NP have similarly high uncertainties on the Physionet data, as the meta-train tasks are very diverse.

ing narrower at 30,000 iterations compared to 4,500 iterations. The non-attentive NP does not have the same problem.

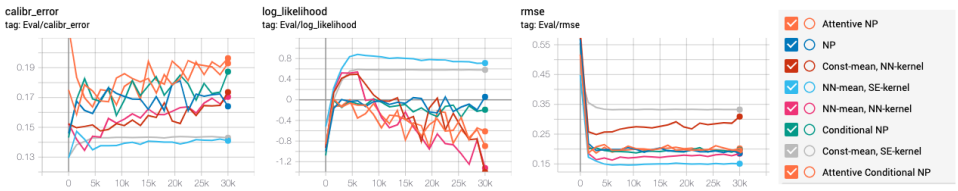
Another factor for this overconfidence is not the model itself, but the data. There is very high similarity among the different meta-train tasks (some Cauchy curves are plotted in Figure 4.2), resulting in overconfidence. On the more heterogenous Physionet meta-train tasks (Figure 4.3), the uncertainty of attentive NPs remains comparably as high as that of the NP (Figure 5.4).

5.3 Overtraining

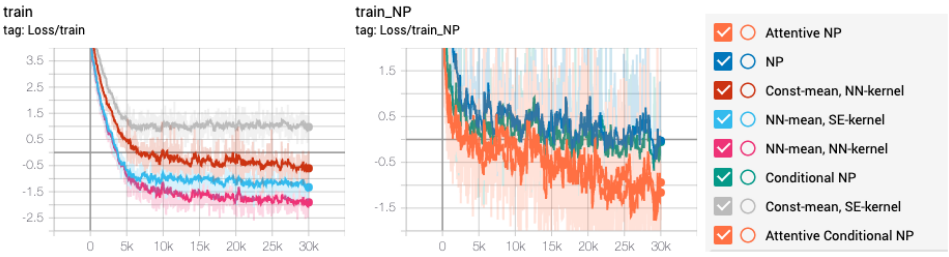
Trying to learn an overly flexible model with many parameters from too little data may result in overtraining, also known as overlearning or overfitting, where the model mistakes artifacts of the meta-train set as actual properties of the underlying distribution that can generalize to the meta-test set. On the other hand, a simple model may be unable to represent the distribution well enough for good regression performance. We find this to be true for the more flexible attentive NNs and the GP models with an NN kernel and/or mean. While Fortuin and Rätsch [2019] claimed that the GP methods cannot overfit on the meta-test context set, we show that they can still overfit on the meta-train dataset.

We study the overtraining phenomenon on the Cauchy dataset by training for 30,000 iterations, past the points where each model converges. As shown by Figure 5.5a, for the more flexible methods, the log-likelihood continues increasing until 5,000 iterations, where it begins to deteriorate. Despite this, the training losses continue to reduce for the most part (Figure 5.5b), indicating overfitting on the meta-training set. The exceptions which are more robust to overfitting are the "const-m, SE-k" GP (in grey), and to a smaller extent, the non-attentive CNP and NP (in dark blue and green) and the "NN-m, SE-k" GP (in light blue).

For the GP models, using a basic SE kernel helps to mitigate this problem. However, the simplest "const-m, SE-k" model results in the worst accuracy in terms of RMSE, because by reverting to the constant mean at the tails, the model



(a) The evaluation metrics, especially log-likelihood, start to deteriorate at around 5,000 iterations for some of the models.



(b) Despite worsening evaluation metrics, training losses mostly continue to improve.

Figure 5.5: Evaluation metrics and training losses of the 8 models over 30,000 iterations.

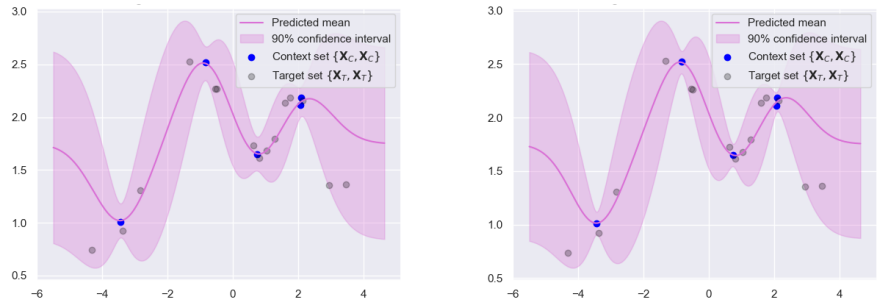
fails to extrapolate well (Figure 5.6a), as discussed previously. Therefore, using an NN mean while keeping the SE kernel appears to be the best in terms of all the metrics, but as we noted in Section 5.1, this is contingent on the SE kernel being a good prior for the dataset in question, the Cauchy dataset.

If we do not have good priors for either the mean or the kernel function, using an NN as both mean and kernel also yields good performance. This means that the models are implicitly learning a mean and kernel function from the data, much like NPs. However, unlike NPs, DKL still involves a base functional kernel that still needs to be hand-picked, and often encodes useful properties such as stationarity or continuity. As such, even an NN kernel can fall back on the base kernel, albeit still with a risk of overtraining the NN (Section 5.3).

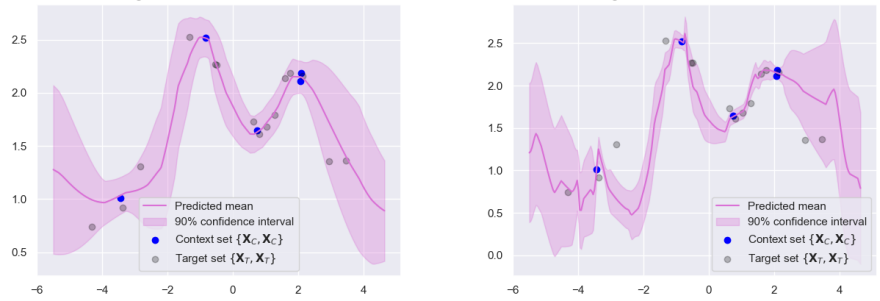
The objective function of the GPs includes a regularization term for the kernel, but not the mean. Despite this, the penalty term does not seem strong enough to regularize the many parameters of an NN kernel, hence the overfitting. The GP models maximize the log marginal likelihood (LML) of each meta-training set $\log p(\boldsymbol{Y}_T|\boldsymbol{X}_T)$, given in closed form by Equation 2.13). As noted by Rasmussen and Williams [2005, Section 5.4.1], the LML has three interpretable terms: a quadratic form in $\boldsymbol{y} - m(\boldsymbol{x})$, a log determinant term, and a term involving $\log 2\pi$. The second term, the log-determinant of the covariance with noise, can be seen as a complexity penalty or regularization term.

There is a trade-off between more flexible models that can fit the data better but are more prone to overfitting, and less flexible models. Nevertheless, we can easily address this in practice by early stopping when we stop seeing an

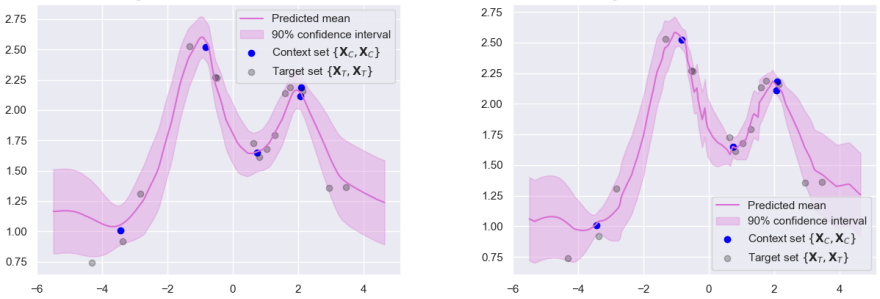
improvement in the log-likelihood on a held-out validation set, or tuning the number of iterations. Alternatively, we could introduce standard regularization techniques for the neural network component(s) such as weight decay.



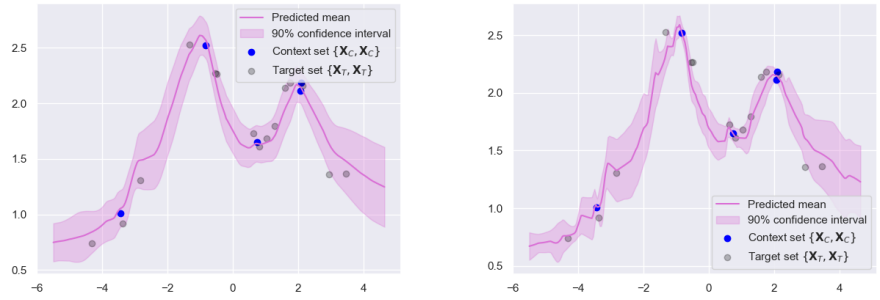
(a) "Con-m, SE-k" GP at 4,500 iterations. (b) "Con-m, SE-k" GP at 30,000 iterations.



(c) "Con-m, NN-k" GP at 4,500 iterations. (d) "Con-m, NN-k" GP at 30,000 iterations.



(e) "NN-m, SE-k" GP at 4,500 iterations. (f) "NN-m, SE-k" GP at 30,000 iterations.



(g) "NN-m, NN-k" GP at 4,500 iterations. (h) "NN-m, NN-k" GP at 30,000 iterations.

Figure 5.6: Overtraining the four GPs variants on a sample meta-test Cauchy function.

Chapter 6

Conclusion

Both NP and meta-learning GP methods are very similar in that they learn the conditional distributions of an underlying stochastic process—approximately in the case of NPs. In this thesis, we evaluate them on identical setups. Especially when using a NN for both the mean and kernel, the GPs approach is very similar to NPs, as they learn implicit mean and kernel functions directly from the data.

Nevertheless, from various experiments on both synthetic and real-world 1D regression data, meta-learning GP methods with either an NN mean and/or kernel have consistently outperformed all NP variants, including attentive and latent variable NPs. Whether the NN mean and/or kernel should be learnt via an NN, depends on our prior knowledge of the data's general properties, and whether that knowledge can be more conveniently encoded as a mean or a kernel function.

Although NPs are attractive for many reasons such as scalability, in practical situations there can be limited meta-training tasks, where such flexible methods are prone to overtraining. A stronger inductive bias via say, an SE kernel or a constant mean, can help prevent this, as shown by our experiments. Furthermore, the GP methods retain the mathematical guarantees of stochastic processes, while the NP methods do not guarantee consistency with respect to some prior process, and as such often have poor calibration, on top of high variance in CE measurements across random seeds.

More real-world datasets can be used to study the applicability of the meta-GPR method. Although most widely used meta-learning datasets are classification tasks Triantafillou et al. [2020], there are some meta-learning regression benchmarks from drug discovery [Tossou et al., 2019] or head pose trajectory estimation Patacchiola et al. [2019].

Bibliography

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems*, pages 41–48, 2007.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[3] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, Jan 2018. ISSN 1095-7200.

[4] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[5] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold Gaussian processes for regression. In *International Joint Conference on Neural Networks*, 2016.

[6] Nicola De Cao, Ivan Titov, and Wilker Aziz. Block neural autoregressive flow. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence*. 2019.

[7] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[8] Francesco Paolo Casale, Adrian V Dalca, Luca Saglietti, Jennifer Listgarten, and Nicolo Fusi. Gaussian process prior variational autoencoders. In *Advances in Neural Information Processing Systems*, 2018.

[9] Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. *CoRR*, 2019. URL http://arxiv.org/abs/1906.02735v5.

[10] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[11] Hyunsun Choi, Eric Jang, and Alexander A. Alemi. WAIC, but Why? Generative Ensembles for Robust Anomaly Detection. *arXiv e-prints*, art. arXiv:1810.01392, October 2018.

[12] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

[13] Andreas C. Damianou and Neil D. Lawrence. Deep Gaussian processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[15] Peter J Diggle and Richard J Gratton. Monte carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):193–212, 1984.

[16] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *CoRR*, 2014.

[17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. In *International Conference on Learning Representations (ICLR)*, 2017.

[18] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In *Advances in Neural Information Processing Systems*, pages 10541–10551, 2019.

[19] Matthew M Dunlop, Mark A Girolami, Andrew M Stuart, and Aretha L Teckentrup. How deep are deep gaussian processes? *The Journal of Machine Learning Research*, 19(1):2100–2145, 2018.

[20] D. Duvenaud, J. R. Lloyd, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302*, 4922., 2013.

[21] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[23] Vincent Fortuin and Gunnar Rätsch. Deep mean functions for meta-learning in gaussian processes. *CoRR*, 2019. URL http://arxiv.org/abs/1901.08098v3.

[24] Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. GP-VAE: Deep probabilistic time series imputation. *arXiv preprint: 1907.04155*, 2019.

[25] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1568–1577, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/franceschi18a.html.

[26] B. J. Frey and N. Jojic. A comparison of algorithms for inference and learning in probabilistic graphical models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27(9):1392–1416, 2005.

[27] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713, 2018a.

[28] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint: 1807.01622*, 2018b.

[29] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

[30] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2016.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[33] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard Turner. Meta-learning probabilistic inference for prediction. 2019a. URL https://openreview.net/forum?id=HkxStoC5F7.

[34] Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional conditional neural processes. *CoRR*, 2019b. URL http://arxiv.org/abs/1910.13556v1.

[35] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *CoRR*, 2018. URL http://arxiv.org/abs/1801.08930v1.

[36] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[37] Thomas L Griffiths, Frederick Callaway, Michael B Chang, Erin Grant, Paul M Krueger, and Falk Lieder. Doing more with less: meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019.

[38] Alexey A. Gritsenko, Jasper Snoek, and Tim Salimans. On the relationship between normalising flows and variational- and denoising autoencoders. In *Deep Generative Models for Highly Structured Data, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HklKEUUY_E.

[39] Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

[41] Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

[42] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[43] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5): 359–366, 1989.

[44] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey. *arXiv e-prints*, art. arXiv:2004.05439, April 2020.

[45] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[46] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087, 2018.

[47] Wenbing Huang, Deli Zhao, Fuchun Sun, Huaping Liu, and Edward Chang. Scalable Gaussian process regression using deep neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[48] Pavel Izmailov, Polina Kirichenko, Marc Finzi, and Andrew Gordon Wilson. Semi-supervised learning with normalizing flows. *CoRR*, 2019.

[49] T. Jebara. *Machine Learning: Discriminative and Generative*. Kluwer Academic, 2004. ISBN 1-4020-7647-9.

[50] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations (ICLR)*, 2019.

[51] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, 2018.

[52] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014.

[53] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[54] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.

[55] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[56] Iryna Korshunova, Jonas Degrave, Ferenc Huszár, Yarin Gal, Arthur Gretton, and Joni Dambre. BRUNO: A deep recurrent model for exchangeable data. In *Advances in Neural Information Processing Systems*, 2018.

[57] Iryna Korshunova, Yarin Gal, Arthur Gretton, and Joni Dambre. Conditional BRUNO: A neural process for exchangeable labelled data. In *Proceedings of the 27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2019.

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[59] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. volume 80 of *Proceedings of Machine Learning Research*, pages 2796–2804, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/kuleshov18a.html.

[60] Brenden Lake, Ruslan Salakhutdinov, and Joshua Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350:1332–1338, 12 2015. doi: 10.1126/science.aab3050.

[61] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *CoRR*, 2016. URL http://arxiv.org/abs/1604.00289v3.

[62] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature Communications*, 10(1):1096, 2019. doi: 10.1038/s41467-019-08987-4. URL https://doi.org/10.1038/s41467-019-08987-4.

[63] Tuan Anh Le, Hyunjik Kim, Marta Garnelo, Dan Rosenbaum, Jonathan Schwarz, and Yee Whye Teh. Empirical evaluation of neural process objectives. In *NeurIPS workshop on Bayesian Deep Learning*, 2018.

[64] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[65] Ke Li and Jitendra Malik. Implicit maximum likelihood estimation. *CoRR*, 2018. URL http://arxiv.org/abs/1809.09087v2.

[66] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727, 2015.

[67] Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The functional neural process. In *Advances in Neural Information Processing Systems*, 2019.

[68] Xuezhe Ma, Xiang Kong, Shanghang Zhang, and Eduard Hovy. Macow: Masked convolutional generative flow. In *Advances in Neural Information Processing Systems*, pages 5891–5900, 2019.

[69] David JC MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

[70] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. 2017.

[71] Shakir Mohamed and Balaji Lakshminarayanan. Learning in Implicit Generative Models. *arXiv e-prints*, art. arXiv:1610.03483, October 2016.

[72] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2013.

[73] Didrik Nielsen and Ole Winther. Closing the Dequantization Gap: PixelCNN as a Single-Layer Flow. *arXiv e-prints*, art. arXiv:2002.02547, February 2020.

[74] Augustus Odena. Semi-Supervised Learning with Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1606.01583, June 2016.

[75] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[76] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[77] Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017.

[78] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[79] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[80] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv e-prints*, art. arXiv:1912.02762, December 2019.

[81] Massimiliano Patacchiola, Jack Turner, Elliot J Crowley, Michael O'Boyle, and Amos Storkey. Deep kernel transfer in gaussian processes for few-shot learning. *arXiv preprint arXiv:1910.05199*, 2019.

[82] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[83] Alec Radford, Karthik Narasimhan, Time Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018.

[84] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[85] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[86] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

[87] Danilo Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

[88] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[89] Jonas Rothfuss, Vincent Fortuin, and Andreas Krause. PACOH: Bayes-Optimal Meta-Learning with PAC-Guarantees. *arXiv e-prints*, art. arXiv:2002.05551, February 2020.

[90] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *CoRR*, 2018. URL http://arxiv.org/abs/1807.05960v3.

[91] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[92] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[93] Eric Schulz, Maarten Speekenbrink, and Andreas Krause. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2018.

[94] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[95] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[96] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

[97] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2005.

[98] Yang Song, Chenlin Meng, and Stefano Ermon. Mintnet: Building invertible neural networks with masked convolutions. *CoRR*, 2019. URL http://arxiv.org/abs/1907.07945v2.

[99] Yunfu Song and Zhijian Ou. Learning neural random fields with inclusive auxiliary generators. *CoRR*, 2018. URL http://arxiv.org/abs/1806.00271v4.

[100] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[101] Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. Functional variational Bayesian neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[102] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[103] Yee Whye Teh. On statistical thinking in deep learning. 2019.

[104] L Theis, A van den Oord, and M Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR 2016)*, pages 1–10, 2016.

[105] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935, 2015.

[106] Prudencio Tossou, Basile Dura, Francois Laviolette, Mario Marchand, and Alexandre Lacoste. Adaptive deep kernel learning. *arXiv preprint arXiv:1905.12131*, 2019.

[107] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgAGAVKPr.

[108] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183, 2013.

[109] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[110] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[111] Mark van der Wilk, Vincent Dutordoir, ST John, Artem Artemev, Vincent Adam, and James Hensman. A Framework for Interdomain and Multioutput Gaussian Processes. *arXiv e-prints*, art. arXiv:2003.01115, March 2020a.

[112] Mark van der Wilk, Vincent Dutordoir, ST John, Artem Artemev, Vincent Adam, and James Hensman. A Framework for Interdomain and Multioutput Gaussian Processes. *arXiv e-prints*, art. arXiv:2003.01115, March 2020b.

[113] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756, 2016.

[114] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[115] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[116] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. Generalizing from a few examples: a survey on few-shot learning. *CoRR*, 2019. URL http://arxiv.org/abs/1904.05046v2.

[117] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 51 of *Proceedings of Machine Learning Research*, pages 370–378. PMLR, 2016.

[118] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*, 2016.

[119] Fei Xu and Joshua B Tenenbaum. Word learning as bayesian inference. *Psychological review*, 114(2):245, 2007.

[120] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. *Show, attend and tell: Neural image caption generation with visual attention*. ICML, 2015.

[121] Wei Ying, Yu Zhang, Junzhou Huang, and Qiang Yang. Transfer learning via learning to transfer. In *International Conference on Machine Learning*, pages 5085–5094, 2018.

[122] John R. Zech, Marcus A. Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric K. Oermann. Confounding variables can degrade generalization performance of radiological deep learning models. *arXiv e-prints*, art. arXiv:1807.00431, July 2018.

[123] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE international conference on computer vision*, pages 1476–1485, 2019.